ORACLES THAT COMPUTE VALUES

STEPHEN FENNER[*], STEVEN HOMER[†], MITSUNORI OGIHARA[‡], AND ALAN SELMAN[§]

Abstract. This paper focuses on complexity classes of partial functions that are computed in polynomial time with oracles in NPMV, the class of all multivalued partial functions that are computable nondeterministically in polynomial time. Concerning deterministic polynomial-time reducibilities, it is shown that

1. A multivalued partial function is polynomial-time computable with k adaptive queries to NPMV if and only if it is polynomial-time computable via $2^k - 1$ nonadaptive queries to NPMV.
2. A characteristic function is polynomial-time computable with k adaptive queries to NPMV if and only if it is polynomial-time computable with k adaptive queries to NP.
3. Unless the Boolean hierarchy collapses, for every k, k adaptive (nonadaptive) queries to NPMV is different than $k+1$ adaptive (nonadaptive) queries to NPMV.

Nondeterministic reducibilities, lowness and the difference hierarchy over NPMV are also studied. The difference hierarchy for partial functions does not collapse unless the Boolean hierarchy collapses, but, surprisingly, the levels of the difference and bounded query hierarchies do not interleave (as is the case for sets) unless the polynomial hierarchy collapses.

Key words. computational complexity, complexity classes, relativized computation, bounded query classes, Boolean hierarchy, multivalued functions, NPMV

AMS subject classifications. 68Q05, 68Q10, 68Q15, 03D10, 03D15

1. Introduction.

In this paper, we study classes of partial functions that can be computed in polynomial time with the help of oracles that are themselves partial functions. We want to know whether there is a difference between computing with function oracles and computing with set oracles. Specifically, we investigate classes of partial functions that can be computed in polynomial time with oracles in NPMV and NPSV, that is, the classes $\mathrm{PF}^{\mathrm{NPMV}}$ and $\mathrm{PF}^{\mathrm{NPSV}}$.

NPMV is the set of all partial multivalued functions that are computed nondeterministically in polynomial time, and NPSV is the set of all partial functions in this class that are single-valued. NPMV captures the complexity of computing witnesses to problems in NP. For example, let *sat* denote the partial multivalued function defined by $sat(x)$ maps to a value y if and only if x encodes a formula of propositional logic and y encodes a satisfying assignment of x. Then, *sat* belongs to NPMV, and the domain of *sat* (i.e., the set of all words x for which the output of $sat(x)$ is non-empty) is the NP-complete satisfiability problem, SAT. Also, NPMV captures the complexity of inverting polynomial time honest functions. To wit, the inverse of every polynomial time honest function belongs to NPMV, and the inverse of every one-one polynomial time honest function belongs to NPSV.

[*] Dept. of Computer Science, University of Southern Maine, Portland, ME 04103. Research partially supported by the National Science Foundation under grant nos. CCR-9209833 and CCR-9501794.

[†] Dept. of Computer Science, Boston University, Boston, MA 01003. Research partially supported by the National Science Foundation under grant nos. CCR-9103055, CCR-9400229, and INT-9123551.

[‡] Dept. of Computer Science, University of Rochester, Rochester, NY 14627. Research done while visiting at the Dept. of Computer Science, SUNY at Buffalo, Buffalo, NY 14260 and while affiliated with Dept. of Computer Science, University of Electro-Communications, Tokyo, Japan, with partial support by the National Science Foundation under grant no. CCR-9002292 and by the NSF-INT-9116781/JSPS-ENGR-207.

[§] Dept. of Computer Science, SUNY at Buffalo, Buffalo, NY 14260. Research partially supported by the National Science Foundation under grant nos. CCR-9002292, INT-9123551, and CCR-9400229.

The class of partial functions with oracles in NP, namely, $\mathrm{PF}^{\mathrm{NP}}$ has been well-studied [13, 1], as have been the corresponding class of partial functions that can be computed nonadaptively with oracles in NP, viz. $\mathrm{PF}_{tt}^{\mathrm{NP}}$ [15], and the classes of partial functions that are obtained by limiting the number of queries to some value $k \geq 1$, namely, $\mathrm{PF}^{\mathrm{NP}[k]}$ and $\mathrm{PF}_{tt}^{\mathrm{NP}[k]}$ [2]. A rich body of results is known about these classes.

Here we raise the question, "What is the difference between computing with an oracle in NPMV versus an oracle in NP?" The answer is not obvious. If the partial function *sat* is provided as an oracle to some polynomial-time computation M, then on a query x, where x encodes a satisfiable formula of propositional logic, the oracle will return some satisfying assignment y. However, if the oracle to M is the NP-compete set SAT, then to this query x, the oracle will only return a Boolean value "yes." On the other hand, by the well-known self-reducibility of SAT, M could compute y for itself by judicious application of a series of adaptive queries to SAT. Indeed Theorem 2.4 states that unbounded access to an oracle in NPMV is no more powerful than such an access to an oracle in NP. However, in Section 3 we will see that the situation for bounded query classes is much more subtle. In general, *function oracles cannot be replaced by set oracles*—but set oracles are still useful. We will show that every partial multivalued function in $\mathrm{PF}^{\mathrm{NPMV}[k]}$ can be computed by a partial multivalued function of the form $f \circ g$, where f is in NPMV and g is a single-valued function belonging to $\mathrm{PF}^{\mathrm{NP}[k]}$. Moreover, most surprisingly, the relationship between access to an oracle in NPMV and access to an oracle in NP is tight regarding set recognition; that is, $\mathrm{P}^{\mathrm{NPMV}[k]} = \mathrm{P}^{\mathrm{NP}[k]}$. This means that when we are computing characteristic functions, k bounded queries to an oracle in NPMV give no more information than the same number of queries to an oracle in NP.

We will show that the levels of the nonadaptive and adaptive bounded query hierarchies interleave (for example, k adaptive queries to a partial function in NPMV is equivalent to $2^k - 1$ nonadaptive queries to a partial function in NPMV), and we will show that these bounded query hierarchies collapse only if the Boolean hierarchy collapses.

In Section 4 we study nondeterministic polynomial time reductions to partial functions in NPMV. Unlike the case for deterministic functions, we will see that just one query to an NP oracle can substitute for an unbounded number of queries to any partial function in NPMV. The hierarchy that is formed by iteratively applying NP reductions is an analogue of the polynomial hierarchy, and we will show that this hierarchy collapses if and only if the polynomial hierarchy collapses.

In Section 5 we will study the difference hierarchy over NPMV. We define $f - g$ to be a partial multivalued function that maps x to y if and only if f maps x to y and g does *not* map x to y, and we define $\mathrm{NPMV}(k) = \{f_1 - (f_2 - (\cdots - f_k)) \mid f_1, \cdots, f_k \in \mathrm{NPMV}\}$. Since the properties of the bounded query hierarchies over NPMV are largely similar to those over NP, one might hope that the same thing happens here—that the difference hierarchy over NPMV and the difference hierarchy over NP are similar. However, the contour of this hierarchy is, to our astonishment, totally different than its analogue for NP. Although $\mathrm{BH} = \bigcup_k \mathrm{NP}(k) \subseteq \mathrm{P}^{\mathrm{NP}}$, with no assumption, we will show that NPMV(2) is included in $\mathrm{PF}^{\mathrm{NPMV}}$ if and only if $\mathrm{PH} = \Delta_2^{\mathrm{P}}$. Also, in this section we will introduce the notion of NPMV-lowness, and we will give a complete characterization of NPMV-lowness.

Consideration of reduction classes with oracles in NPSV, to be studied in Section 6, is motivated in part by a desire to understand how difficult it is to compute satisfying assignments for satisfiable formulas. We take the point of view that a partial multivalued function is easy to compute if for each input string in the domain of the function, some value of the function is easy to compute. For this reason, we define the following technical notions. Given partial multivalued functions f and g, define g to be a *refinement* of f if $dom(g) = dom(f)$ and for all $x \in dom(g)$ and all y, if y is a value of $g(x)$, then y is a value of $f(x)$. Let $\mathcal{F}$ and $\mathcal{G}$ be classes of partial multivalued functions. Purely as a convention, if f is a partial multivalued function, we define $f \in_c \mathcal{G}$ if $\mathcal{G}$ contains a refinement g of f, and we define $\mathcal{F} \subseteq_c \mathcal{G}$ if for every $f \in \mathcal{F}$, $f \in_c \mathcal{G}$. This notation is consistent with our intuition that $\mathcal{F} \subseteq_c \mathcal{G}$ should entail that the complexity $\mathcal{F}$ is not greater than the complexity of $\mathcal{G}$. Let PF denote the class of partial functions that are computable deterministically in polynomial time. The assertion "NPMV $\subseteq_c$ PF" means that every partial multivalued function in NPMV has a refinement that can be computed efficiently by some deterministic polynomial time transducer. It is well-known that $sat \in_c$ PF if and only if NPMV $\subseteq_c$ PF if and only if P = NP [15]. Thus, one does not expect that $sat \in_c$ PF. Is sat computable in some larger single-valued class of partial functions? Selman [15] showed that PF $\subseteq$ NPSV $\subseteq$ $\mathrm{PF}_{tt}^{\mathrm{NP}}$. If $sat \in_c$ NPSV, then the polynomial hierarchy collapses [11], and it is an open question whether $sat \in_c$ NPSV or whether $sat \in_c \mathrm{PF}_{tt}^{\mathrm{NP}}$. (Watanabe and Toda [18] have shown that $sat \in_c \mathrm{PF}_{tt}^{\mathrm{NP}}$ relative to a random oracle.) We will consider classes of the form $\mathrm{PF}^{\mathrm{NPSV}[k]}$ and $\mathrm{PF}_{tt}^{\mathrm{NPSV}[k]}$, where $k \geq 1$, and we will show that the adaptive and the nonadaptive classes form proper hierarchies unless the Boolean hierarchy collapses. Thus, these classes form a finer classification in which to study the central question of whether sat has a refinement in some interesting class of single-valued partial functions.

Finally, we note in passing that the complexity theory of decision problems, i.e., of sets, is extremely well developed. Although the computational problems in which we are most interested are naturally thought of as partial multivalued functions, the structural theory to support classification of these problems has been slight. By introducing several natural hierarchies of complexity classes of partial multivalued functions, with strong evidence supporting these claims, we intend this paper to make significant steps in correcting this situation.

2. Preliminaries. We fix Σ to be the finite alphabet $\{0,1\}$. $<$ denotes the standard canonical lexicographic order on Σ^*. Let $f : \Sigma^* \mapsto \Sigma^*$ be a partial multivalued function. We write $f(x) \mapsto y$ (or, $f(x)$ maps to y), if y is a value of f on input string x. Define $graph(f) = \{\langle x, y\rangle \mid f(x) \mapsto y\}$, $dom(f) = \{x \mid \exists y(f(x) \mapsto y)\}$, and $range(f) = \{y \mid \exists x(f(x) \mapsto y)\}$. We will say that f is undefined at x if $x \notin dom(f)$.

A transducer T is a nondeterministic Turing machine with a read-only input tape, a write-only output tape, and accepting states in the usual manner. A transducer T computes a value y on an input string x if there is an accepting computation of T on x for which y is the final content of T's output tape. (In this case, we will write $T(x) \mapsto y$.) Such transducers compute partial, multivalued functions. (As transducers do not typically accept all input strings, when we write "function", "partial function" is always intended. If a function f is total, it will always be explicitly noted.)

- NPMV is the set of all partial, multivalued functions computed by nondeterministic polynomial time-bounded transducers;
- NPSV is the set of all $f \in$ NPMV that are single-valued;
- PF is the set of all partial functions computed by deterministic polynomial time-bounded transducers.

A function f belongs to NPMV if and only if it is polynomially length-bounded and $graph(f)$ belongs to NP. The domain of every function in NPMV belongs to NP.

These definitions originate in Book, Long, and Selman's study of restricted-access relativizations [5].

Now we describe oracle Turing machines with oracles that compute partial functions. For the moment, we assume that the oracle is a single-valued partial function. Let $\perp$ be a symbol not belonging to the finite alphabet Σ. In order for a machine M to access a partial function oracle, M contains a write-only input oracle tape, a separate read-only output oracle tape, and a special oracle call state q. When M enters state q, if the string currently on the oracle input tape belongs to the domain of the oracle partial function, then the result of applying the oracle appears on the oracle output tape, and if the string currently on the oracle input tape does not belong to the domain of the oracle partial function, then the symbol $\perp$ appears on the oracle output tape. Thus, if the oracle is some partial function g, given an input x to the oracle, the oracle, if called, returns a value $g(x)$ if one exists, and returns $\perp$ otherwise. (It is possible that M may read only a portion of the oracle's output if the oracle's output is too long to read with the resources of M.) We shall assume, without loss of generality, that M never makes the same oracle query more than once, i.e., all of M's queries (on any possible computation path) are distinct. $\mathrm{PF}^{\mathrm{NP}}$ is the class of partial functions computed in polynomial time with oracles in NP. $\mathrm{PF}_{tt}^{\mathrm{NP}}$ is the class of partial functions that can be computed nonadaptively with oracles in NP; that is, a partial function f is in $\mathrm{PF}_{tt}^{\mathrm{NP}}$ if there is a deterministic oracle Turing machine transducer T such that $f \in \mathrm{PF}^{\mathrm{NP}}$ via T with an oracle L in NP and a total polynomial time computable function $g : \{0,1\}^* \mapsto (c\{0,1\}^*)^*$ such that, for each input x to T, T only makes queries to L from the list $g(x)$.

If g is a single-valued partial function and M is a deterministic oracle transducer as just described, then we let $M[g]$ denote the single-valued partial function computed by M with oracle g.

Definition 2.1. *Let f and g be multivalued partial functions.* f is Turing reducible to g in polynomial time, $f \leq_{\mathrm{T}}^{\mathrm{P}} g$, *if for some deterministic oracle transducer M, for every single-valued refinement g' of g, $M[g']$ is a single-valued refinement of f.*[1]

Proposition 2.1. *Polynomial time Turing reducibility, $\leq_{\mathrm{T}}^{\mathrm{P}}$, is a reflexive and transitive relation over the class of all partial multivalued functions.*

Let $\mathcal{F}$ be a class of partial multivalued functions. $\mathrm{PF}^{\mathcal{F}}$ denotes the class of partial multivalued functions f that are $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible to some $g \in \mathcal{F}$. $\mathrm{PF}^{\mathcal{F}[k]}$ (respectively, $\mathrm{PF}^{\mathcal{F}[\log]}$) denotes the class of partial multivalued functions f that are $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible to some $g \in \mathcal{F}$ via a machine that, on input x, makes k adaptive queries (respectively, $\mathcal{O}(\log |x|)$ adaptive queries) to its oracle.

$\mathrm{PF}_{tt}^{\mathcal{F}}$ denotes the class of partial multivalued functions f that are $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible to some $g \in \mathcal{F}$ via an oracle Turing machine transducer that queries its oracle nonadaptively. That is, a partial multivalued function f is in $\mathrm{PF}_{tt}^{\mathcal{F}}$ if there is an oracle

[1] A notion of polynomial-time Turing reducibility between partial functions is defined by Selman [15]. It is important to note that the definition given here is *different* than the one given there. Here the oracle "knows" when a query is not in its domain. In the earlier definition, this is not the case. The authors recommend that the reducibility defined in the earlier paper should in the future be denoted as $\leq_{\mathrm{T}}^{\mathrm{PP}}$, which is the common notation for reductions between promise problems. We make this recommendation because conceptually and technically this reducibility between functions is equivalent to a promise problem reduction. Also, we note that the reducibility defined by Selman [15] is not useful for our purposes here. In particular, it is easy to see that iterating reductions between functions in NPMV does not gain anything new unless the oracle is endowed with the ability to know its domain.

Turing machine transducer T such that $f \in \mathrm{PF}^{\mathcal{F}}$ via T with an oracle g in $\mathcal{F}$ and a polynomial time computable function $h : \{0,1\}^* \mapsto (c\{0,1\}^*)^*$ such that, for each input x to T, T only calls the oracle g on strings in the list $h(x)$.

$\mathrm{PF}_{tt}^{\mathcal{F}[k]}$ denotes the class of partial multivalued functions f that are $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible to some $g \in \mathcal{F}$ via a machine that makes k nonadaptive queries to its oracle, i.e., just as in the last paragraph, but with $h : \{0,1\}^* \mapsto (c\{0,1\}^*)^k$.

$\mathrm{P}^{\mathcal{F}}$, $\mathrm{P}^{\mathcal{F}[k]}$, $\mathrm{P}^{\mathcal{F}[\log]}$, $\mathrm{P}_{tt}^{\mathcal{F}}$ and $\mathrm{P}_{tt}^{\mathcal{F}[k]}$, respectively, denote the classes of all characteristic functions contained in $\mathrm{PF}^{\mathcal{F}}$, $\mathrm{PF}^{\mathcal{F}[k]}$, $\mathrm{PF}^{\mathcal{F}[\log]}$, $\mathrm{PF}_{tt}^{\mathcal{F}}$ and $\mathrm{PF}_{tt}^{\mathcal{F}[k]}$.

For a class of sets $\mathcal{C}$, we may say that $\mathrm{PF}^{\mathcal{C}}$ denotes the class of partial multivalued functions that are $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible to the characteristic function of some set in $\mathcal{C}$. $\mathrm{PF}^{\mathcal{C}[k]}$, $\mathrm{PF}^{\mathcal{C}[\log]}$, $\mathrm{PF}_{tt}^{\mathcal{C}}$, $\mathrm{PF}_{tt}^{\mathcal{C}[k]}$, $\mathrm{P}^{\mathcal{C}}$, $\mathrm{P}^{\mathcal{C}[k]}$, $\mathrm{P}^{\mathcal{C}[\log]}$, $\mathrm{P}_{tt}^{\mathcal{C}}$, and $\mathrm{P}_{tt}^{\mathcal{C}[k]}$ are defined similarly. In particular, $\mathrm{PF}^{\mathrm{NP}}$ is the class of partial multivalued functions computed in polynomial time with oracles in NP, and $\mathrm{PF}_{tt}^{\mathrm{NP}}$ is the class of partial functions that can be computed nonadaptively with oracles in NP. In the current literature, these classes contain single-valued functions only. The reason is that heretofore, polynomial time Turing reducibility, $\leq_{\mathrm{T}}^{\mathrm{P}}$, has been defined as a binary relation over single-valued objects. To see that $\mathrm{PF}^{\mathrm{NP}}$ contains partial functions that are not single-valued, consider the partial single-valued function *maxsat* that on an input x where x encodes a formula of propositional logic, maps to the encoding of the lexicographically largest satisfying assignment of x, if $x \in \mathrm{SAT}$. Clearly, $\mathit{maxsat} \in \mathrm{PF}^{\mathrm{NP}}$, and $\mathit{sat} \leq_{\mathrm{T}}^{\mathrm{P}} \mathit{maxsat}$ by Definition 2.1, so the partial multivalued function *sat* belongs to $\mathrm{PF}^{\mathrm{NP}}$. Readers are free to interpret references to $\mathrm{PF}^{\mathrm{NP}}$ and $\mathrm{PF}_{tt}^{\mathrm{NP}}$ with their familiar meaning because the results that we will state for these classes, and for the corresponding bounded query classes, remain correct if the classes are replaced with the result of including only the single-valued partial functions that they contain.

Given a class of partial multivalued functions $\mathcal{F}$, let $\mathcal{F}/_{sv}$ denote the class of single-valued partial functions that $\mathcal{F}$ contains.

All the classes of partial multivalued functions that we have defined, other than NPMV, are closed "backwards" under refinement. That is, with the exception of NPMV, the following Property 1 holds for each of these classes $\mathcal{F}$:

$$(1) \qquad f \in \mathcal{F} \wedge f \text{ is a refinement of } g \rightarrow g \in \mathcal{F}$$

Let us say that classes that satisfy property 1 are *c-closed.* The c-closure of a class is c-closed. Let us say that a *basis* for a class $\mathcal{F}$ is a subset $\mathcal{F}'$ of $\mathcal{F}$ such that for all $f \in \mathcal{F}$, there is an $f' \in \mathcal{F}'$ such that f' is a refinement of f. Essentially all interesting c-closed classes are uncountable, but this is not problematic because they all arise as the c-closure of classes that are countable and effectively enumerable (that is, they are indexed by machines of some appropriate type). Property 2 holds for every class of partial functions $\mathcal{F}$ that is Turing reducible in polynomial time to a class of single-valued partial functions.

$$(2) \qquad f \in \mathcal{F} \rightarrow \exists f'[f' \text{ is a single-valued refinement of } f \wedge f' \in \mathcal{F}]$$

For example, Property 2 holds for $\mathrm{PF}^{\mathrm{NP}}$. Property 2 states that the set of single-valued functions in $\mathcal{F}$ is a basis for $\mathcal{F}$. (To use an analogy from lattice theory, if one thinks of the single-valued functions as "atoms," then property 2 is the "atomic basis property."). Also, note that "is a refinement of" is reflexive and transitive over the class of all partial multivalued functions.

PROPOSITION 2.2. *If $\mathcal{F}$ satisfies property 1, then $g \in_c \mathcal{F} \leftrightarrow g \in \mathcal{F}$ and $\mathcal{G} \subseteq_c \mathcal{F} \leftrightarrow \mathcal{G} \subseteq \mathcal{F}$.*

Thus, $\in_c$ is identical to class containment and $\subseteq_c$ is identical to class inclusion for the classes we have defined.

PROPOSITION 2.3. *If $\mathcal{F}$ satisfies property 2 and $\mathcal{G}$ satisfies property 1, then $\mathcal{F}/_{sv} \subseteq \mathcal{G}/_{sv} \leftrightarrow \mathcal{F} \subseteq \mathcal{G}$.*

Beigel [2] proved that for all $k \geq 1$, $\mathrm{PF}^{\mathrm{NP}[k]}/_{sv} \subseteq \mathrm{PF}_{tt}^{\mathrm{NP}[2^k-1]}/_{sv}$. Using Proposition 2.3, it follows that $\mathrm{PF}^{\mathrm{NP}[k]} \subseteq \mathrm{PF}_{tt}^{\mathrm{NP}[2^k-1]}$. This example illustrates that known inclusion results for the classes we are considering remain true under the new interpretation that these classes contain multivalued functions. Thus, passing to multivalued functions does not disturb our current understanding of previously studied function classes. We are recasting the definitions in no small part because we will be dealing with many classes that (most likely) do not satisfy property 2, and hence our results are strictly more general.

Obviously $\mathrm{PF}^{\mathrm{NP}} \subseteq \mathrm{PF}^{\mathrm{NPMV}}$. Conversely, for a function $f \in \mathrm{NPMV}$, define f' to be a function such that $f'(x) = \min\{y \mid f(x) \mapsto y\}$. The function f' is a single-valued refinement of f and in $\mathrm{PF}^{\mathrm{NP}}$, so $\mathrm{NPMV} \subseteq \mathrm{PF}^{\mathrm{NP}}$ by Proposition 2.2. This implies that $\mathrm{PF}^{\mathrm{NPMV}} \subseteq \mathrm{PF}^{\mathrm{PF}^{\mathrm{NP}}} = \mathrm{PF}^{\mathrm{NP}}$ since $\leq_{\mathrm{T}}^{\mathrm{P}}$ is transitive. Therefore, the following theorem holds.

THEOREM 2.4. $\mathrm{PF}^{\mathrm{NPMV}} = \mathrm{PF}^{\mathrm{NP}}$.

Theorem 2.4 states that unbounded access to an oracle in NPMV is no more powerful than such an access to an oracle in NP.

The following examples, the first of which was pointed out by Buhrman [3], illustrate the power of $\mathrm{PF}^{\mathrm{NPMV}}$ and $\mathrm{PF}_{tt}^{\mathrm{NPMV}}$. Consider the partial multivalued function *maxTsat* defined as follows:

> $maxTsat(x) \mapsto y$, if y is a satisfying assignment of x with the maximum number of *true*'s.

Obviously, *maxTsat* belongs to $\mathrm{PF}^{\mathrm{NPMV}}$. Let f be a function that maps a pair (x, n) to y if and only if y is a satisfying assignment of x with n *true*'s. Since the number of variables in a formula is bounded by its length, it holds that $maxTsat(x) = f(x, n_x)$, where n_x is the largest n, $1 \leq n \leq |x|$ such that $(x,n) \in dom(f)$. This implies that $maxTsat \in \mathrm{PF}_{tt}^{\mathrm{NPMV}}$.

Similarly, the partial multivalued function *maxclique*, that on input a graph G outputs a clique of maximum size, belongs to $\mathrm{PF}_{tt}^{\mathrm{NPMV}}$. The function *MaxEdgeWeightClique* that is defined over edge-weighted graphs and that outputs a clique of maximum weight, if G has a clique, belongs to $\mathrm{PF}^{\mathrm{NPMV}}$, but may not belong to $\mathrm{PF}_{tt}^{\mathrm{NPMV}}$ because weights may grow exponentially.

We should note that several of the classes we investigate here seem to capture the complexity of finding witnesses to NP-optimization problems. This observation is explored by Chen and Toda [9] and by Wareham [17].

3. Bounded Query Classes. In this section we prove a number of basic results clarifying the structure of the bounded adaptive and nonadaptive query hierarchies over NPMV, both for computing functions and for set recognition. The new hierarchies are mostly analogous to those over NP, but there are some interesting and subtle differences. General techniques developed in this section are reminiscent of the "mind-change" technique [2, 19]. We will use them first to compare $\mathrm{PF}^{\mathrm{NPMV}[k]}$ and $\mathrm{PF}_{tt}^{\mathrm{NPMV}[k]}$ with $\mathrm{PF}^{\mathrm{NP}[k]}$ and $\mathrm{PF}_{tt}^{\mathrm{NP}[k]}$, respectively.

The following two propositions are central to the rest of the paper and will be used in several places later on: Theorems 3.3, 3.5, 3.7, and 3.8, Lemma 5.10, and Proposition 6.6. Each proposition abstracts the general idea, common to all these results, that we can replace the oracle queries in any $\mathrm{PF}^{\mathrm{NPMV}}$ computation by nondeterminism in a way that preserves information about outputs of the computation. Proposition 3.1 deals with computations making adaptive queries; Proposition 3.2 deals with nonadaptive queries. After we prove them we will discuss briefly how they are used.

PROPOSITION 3.1. *Let $t \in \mathrm{PF}$. Let f be in $\mathrm{PF}^{\mathrm{NPMV}}$ be computed by a deterministic oracle Turing machine transducer M with $g \in \mathrm{NPMV}$ as the oracle. Suppose that M on x makes $t(x)$ queries to its oracle. Then there is an* NPMV *function $s[M,g] : \Sigma^* \times \Sigma^* \mapsto \Sigma^*$ that satisfies the following conditions.*

1. *$\lambda x.[s[M,g](x, 0^{t(x)})]$ is total, single-valued and polynomial time computable.*
2. *For every x, there uniquely exists $a_x \in \Sigma^{t(x)}$ such that*
   (a) *for every $a \in \Sigma^{t(x)}$, $(x,a) \in dom(s[M,g])$ if and only if $a \leq a_x$,*
   (b) *$f(x)$ is undefined if and only if $s[M,g](x,a_x)$ maps to 0, and*
   (c) *for every y, if $s[M,g](x,a_x)$ maps to $1y$, then $f(x)$ maps to y.*

Proof. Let t, f, M and g be as in the hypothesis. The idea of the proof is as follows: given an input x, say that a string $a \in \Sigma^{t(x)}$ is *okay* if there is a legitimate computation path of $M(x)$ where the ith query is in $dom(g)$ if and only if the ith bit of a is 1. The "magic" string a_x that we seek will be the lexicographically maximum okay string. Computing the function $s[M,g]$ involves, among other things, guessing if there is an okay string no smaller than the given input string $a \in \Sigma^{t(x)}$. This must be done indirectly, as one cannot necessarily verify—even nondeterministically—whether a given string is okay.

Let N be a polynomial time nondeterministic Turing machine witnessing that $g \in \mathrm{NPMV}$. Define U to be the following machine: On input x and $b \in \Sigma^{t(x)}$, U simulates M on input x in the following manner:

- For each $i, 1 \leq i \leq t(x)$, when M makes its i-th query q_i, U behaves as follows:
  - If the i-th symbol in b is a 0, then U assumes that the answer is $\perp$.
  - If the i-th symbol in b is a 1, then U simulates N on q_i. If N on q_i does not accept, then U halts without accepting, and if N on q_i outputs some z_i, then U assumes that the answer is z_i.
- When M enters a halting state, U behaves as follows:
  - If M rejects, then U outputs 0.
  - If M outputs y, then U outputs $1y$.

Let r be the NPMV function defined by U. For every x, U on $(x, 0^{t(x)})$ makes no nondeterministic guesses. So, U on $(x, 0^{t(x)})$ always has a unique output and $\lambda x.[r(x, 0^{t(x)})]$ is polynomial time computable.

For a given x, let b_x be the largest $b \in \Sigma^{t(x)}$ such that U on (x,b) has an output, and let π be an arbitrary computation path of U on (x, b_x) that leads to an output. Suppose that along path π, U generates query strings $q_1, \cdots, q_{t(x)}$ in this order and computes the answers to them as $z_1, \cdots, z_{t(x)}$, respectively. By definition, for every i such that $z_i \neq \perp$, $g(q_i) \mapsto z_i$. Furthermore, we claim that for every i such that $z_i = \perp$, $q_i \notin dom(g)$. This is seen as follows: Assume that there is some i such that $z_i = \perp$ and $q_i \in dom(g)$. Let j be the smallest such i. By the minimality of j, there exist some c and some computation path π' of U on (x,c) such that along path π',

(i) U has an output,
(ii) the first j queries U computes are $q_1, \cdots, q_j$,
(iii) the first $j-1$ answers U computes are $z_1, \cdots, z_{j-1}$,
(iv) the j-th answer U computes is not $\perp$, and

(v) π and π' agree until q_j.

Let $b_x = u0w$ with $|u| = j-1$. By (ii), (iii) and (iv), we have $c = u1v$ for some v. Thus, $c > b_x$. By (i), U on (x, c) has an output. So, by the maximality of b_x, $b_x \geq c$, which contradicts $c > b_x$. Therefore, for every i such that $z_i = \bot$, $q_i \notin \mathit{dom}(g)$.

Thus we see that all the answers $z_1, \cdots, z_{t(x)}$ are correct. Define g' to be a single-valued refinement of g that is defined by path π. U on (x, b_x) along path π correctly simulates $M[g']$ on x. Thus, it holds that

$$\begin{array}{rcl} x \in \mathit{dom}(f) & \leftrightarrow & M[g'] \text{ has an output} \\ & \leftrightarrow & U \text{ on } (x, b_x) \text{ along path } \pi \text{ outputs a string of the form } 1y, \text{ and} \\ x \notin \mathit{dom}(f) & \leftrightarrow & M[g'] \text{ does not have an output} \\ & \leftrightarrow & U \text{ on } (x, b_x) \text{ along path } \pi \text{ outputs } 0. \end{array}$$

Therefore, U on (x, b_x) along path π outputs 0 if and only if $f(x)$ is undefined, and if U on (x, b_x) along path π outputs $1y$ then $f(x)$ maps to y.

Now define V to be the machine that, on input (x, a) with $|a| = t(x)$, behaves as follows:

- if $a = 0^{t(x)}$, then V simulates U on (x, a), and
- if $a \neq 0^{t(x)}$, then V guesses $b \in \Sigma^{t(x)}$ with $b \geq a$ and simulates U on (x, b).

Let s be the NPMV function defined by V. We claim that s is the desired function. Since V and U are the same on input $(x, 0^{t(x)})$, $\lambda x.[s(x, 0^{t(x)})]$ is total, single-valued and polynomial time computable. Let a_x be the largest $a \in \Sigma^{t(x)}$ such that $(x, a) \in \mathit{dom}(s)$. It is not hard to see that $a_x = b_x$. Since b_x is the largest b such that $(x, b) \in \mathit{dom}(r)$, and V on (x, a) simulates U on (x, b) for all $b \geq a$ except when $a = 0^{t(x)}$, it holds that

(i) for every $a > a_x$, $(x, a) \notin \mathit{dom}(s)$,
(ii) for every $a \leq a_x$, $(x, a) \in \mathit{dom}(s)$,
(iii) $x \notin \mathit{dom}(f)$ if and only if $s(x, a_x) \mapsto 0$, and
(iv) if $s(x, a_x) \mapsto 1y$, then $f(x) \mapsto y$.

Hence all the required properties are satisfied. This proves the proposition. □

PROPOSITION 3.2. *Let $t \in$ PF. Let f in $\mathrm{PF}_{tt}^{\mathrm{NPMV}}$ be computed by a deterministic oracle Turing machine transducer M with $g \in$ NPMV as the oracle. Suppose that M on x makes $t(x)$ queries. Then there is an* NPMV *function $s[M, g] : \Sigma^* \times \{0, \cdots, t(x)\} \mapsto \Sigma^*$ that satisfies the following conditions:*

1. *$\lambda x.[s[M, g](x, 0)]$ is total, single-valued and polynomial time computable,*
2. *for every x and $0 \leq m \leq n \leq t(x)$, if $(x, n) \in \mathit{dom}(s[M, g])$ then $(x, m) \in \mathit{dom}(s[M, g])$,*
3. *for every x, $f(x)$ is undefined if and only if $s[M, g](x, n_x)$ maps to 0, and*
4. *for every x and y, $f(x)$ maps to y if and only if $s[M, g](x, n_x)$ maps to $1y$,*

where n_x is the largest $n \in \{0, \cdots, t(x)\}$ such that $(x, n) \in \mathit{dom}(s[M, g])$.

Proof. Let t, f, M, and g be as in the hypothesis and let N be a nondeterministic Turing machine witnessing that $g \in$ NPMV. The idea of this proof is analogous to, but simpler than, that of the last proposition. The "magic" number n_x that we seek will be the number of queries of $M(x)$ that are in $\mathit{dom}(g)$.

Let h be the function defined by the following machine U: On input x and $n \leq t(x)$, U behaves as follows:

(A) U first computes all the query strings $q_1, \cdots, q_{t(x)}$ of M on x.

(B) If $n = 0$, then for every i, U assumes that the answer to q_i is $\bot$. If $n > 0$, then U does the following:

- For each i, U simulates N on q_i. If N does not accept q_i, then U assumes that the answer to q_i is $\perp$, and if N outputs w on q_i, then U assumes that the answer to q_i is w. After doing this, if the number of answers obtained as $\perp$ is larger than $t(x) - n$, then U halts without accepting.

(C) U simulates M on x using the answers computed in (B). If M rejects, then U outputs 0, and if M outputs z, then U outputs $1z$.

We claim that h is the desired function.

For every x, U on $(x, 0)$ runs deterministically and always has an output. So, $\lambda x.[h(x, 0)]$ is total, single-valued, and polynomial time computable. Suppose $0 < m \leq n \leq t(x)$ and $(x, n) \in \mathit{dom}(h)$. Then U must have an accepting path in step (B), where it obtains at most $t(x) - n$ query answers as $\perp$. The same set of query answers will also allow U to accept on input (x, m).

For each x, let n_x be the maximum n such that $(x, n) \in \mathit{dom}(h)$. For every x, n_x coincides with the number of queries of M on x that are in $\mathit{dom}(g)$. Let π be any computation path of U on (x, n_x) leading to an output. Let $z_1, \cdots, z_{t(x)}$ be the answers that U computes along path π for queries $q_1, \cdots, q_{t(x)}$, respectively. Then, by the maximality of n_x, for every i, $z_i = \perp$ if and only if $q_i \notin \mathit{dom}(g)$ and if $z_i \neq \perp$, then $g(q_i)$ maps to z_i. So, the output along path π is 0 if and only if $f(x)$ is undefined, and if the output is $1y$, then $f(x)$ maps to y. Therefore, h is the desired function. □

We will use Propositions 3.1 and 3.2 in three ways. First, we can simulate the behavior of a $\mathrm{PF}^{\mathrm{NPMV}}$ (respectively $\mathrm{PF}_{tt}^{\mathrm{NPMV}}$) computation on input x purely nondeterministically, *provided* we know a_x (respectively n_x). Such a simulation $s[M, g]$ always accepts, tells us whether $M(x)$ outputs a value, and if so, provides us with an output. Second, $\mathit{dom}(s[M, g])$ is such that we can find a_x (resp. n_x) by binary search with an NP oracle. Third, the fact that $s[M, g](x, 0^{t(x)})$ (resp. $s[M, g](x, 0)$) can be computed deterministically saves us an NP query so that we can get an exact connection between bounded adaptive and nonadaptive NPMV queries in Theorem 3.5.

Let f and g be partial multivalued functions. $f \circ g$ denotes the function h such that for every x,

- $h(x)$ maps to y if and only if there exists some z such that $g(x)$ maps to z and $f(z)$ maps to y.

Let $\mathcal{F}$ and $\mathcal{G}$ be classes of partial multivalued functions. $\mathcal{F} \circ \mathcal{G}$ denotes $\{f \circ g \mid f \in \mathcal{F}$ and $g \in \mathcal{G}\}$.

Although composition is a natural operator and an important tool in our investigations, we should caution that the classes we consider tend not to be closed under composition, and the composition of two easy-to-compute functions may be very difficult. To see this, consider the functions r and s defined as follows: $r(x) \mapsto 1$, for all $x \neq 0$, and r is undefined at $x = 0$; $s(x) \mapsto 0$, for all strings x, and $s(x) \mapsto 1$, if $x \in K$, where K is a complete recursively enumerable set. The partial multivalued functions r and s have refinements in PF, but $\mathit{dom}(r \circ s) = K$, so $r \circ s$ does *not* have a refinement in PF.

The following theorem relates computing with an oracle in $\mathrm{NPMV}[k]$ to computing with an oracle in $\mathrm{NP}[k]$. In particular, we see that every partial multivalued function in $\mathrm{PF}^{\mathrm{NPMV}[k]}$ can be computed by a partial multivalued function of the form $f \circ g$, where f is in NPMV and g is a single-valued function in $\mathrm{PF}^{\mathrm{NP}[k]}$.

THEOREM 3.3.

(1) *For every* $k \geq 1$, $\mathrm{PF}^{\mathrm{NPMV}[k]} \subseteq_c \mathrm{NPMV} \circ (\mathrm{PF}^{\mathrm{NP}[k]}/_{sv})$.

(2) *For every $k \geq 1$, $\mathrm{PF}^{\mathrm{NPMV}[k]} \subseteq \mathrm{NPMV} \circ \mathrm{PF}^{\mathrm{NP}[k]}$.*

Proof. Let $f \in \mathrm{PF}^{\mathrm{NPMV}[k]}$ via a deterministic oracle Turing machine transducer M with $g \in \mathrm{NPMV}$ as the oracle. Let $h = s[M, g]$ be the function defined in Proposition 3.1. Let V be a machine witnessing that $h \in \mathrm{NPMV}$. Define b to be a function that maps x to a_x, where a_x is the largest $a \in \Sigma^k$ such that $(x, a) \in \mathit{dom}(h)$. Recall that $b(x)$ is defined for every x. A binary search algorithm over $\Sigma^k - \{0^k\}$ computes b in polynomial time with oracle $\mathit{dom}(h)$. The number of questions is exactly k, so, $b \in \mathrm{PF}^{\mathrm{NP}[k]}/_{sv}$. Now define f' to be a function that maps x to $(x, b(x))$. Define V' to be a machine that on input (x, a) simulates V on (x, a) and does not accept if either V does not accept or V outputs 0, and outputs y if V outputs $1y$. Let h' be the partial multivalued function defined by V'. Then $h' \in \mathrm{NPMV}$.

Now define $r(x) = h'(f'(x))$. It is easy to see that $r(x)$ is undefined if and only if $f(x)$ is undefined, and if $r(x)$ maps to y, then $f(x)$ maps to y. Therefore, r is a refinement of f and is in $\mathrm{NPMV} \circ \mathrm{PF}^{\mathrm{NP}[k]}/_{sv}$. This proves (1).

To prove (2) we proceed exactly as above except that instead of f' we use a new function f'' defined so that for all x and y, $f''(x)$ maps to (x, y) if and only if either

1. $y = b(x)$, or
2. $y = 0^k 1 z$ for some z such that $f(x)$ maps to z.

Note that f' is a refinement of f'', so $f'' \in \mathrm{PF}^{\mathrm{NP}[k]}$ by Proposition 2.2. Define V'' to be a machine that on input (x, a) first checks if a is of the form $0^k 1 z$ for some z. If so, V'' outputs z and halts. Otherwise, V'' behaves exactly as V' above. Let h'' be the function defined by V''. We have $h'' \in \mathrm{NPMV}$ as before. Now defining $r'(x) = h''(f''(x))$, we show that $r' = f$, completing the proof.

Suppose $f(x)$ maps to z. Then $f''(x)$ maps to $(x, 0^k 1 z)$ and $h''(x, 0^k 1 z)$ maps to z, so $r'(x)$ maps to z. Conversely, suppose $r'(x)$ maps to z. Then it must be the case that either $f''(x)$ maps to $(x, b(x))$ and $h''(x, b(x))$ maps to z, or $f''(x)$ maps to $(x, 0^k 1 z)$ and $h''(x, 0^k 1 z)$ maps to z. In the latter case, $f(x)$ maps to z by the definition of f''. In the former case, $V(x, b(x))$ must output $1z$, and thus $f(x)$ maps to z. □

Selman [15] showed that $\mathrm{PF}^{\mathrm{NP}[\log]} \neq \mathrm{PF}_{tt}^{\mathrm{NP}}$ unless $\mathrm{P} = \mathrm{FewP}$ and $\mathrm{R} = \mathrm{NP}$.[2] The next two theorems are interesting because they imply (i) that composing on the left with NPMV is enough to absorb the difference between the two reduction classes, and (ii) that the NPMV analog of Selman's result is *false*.

THEOREM 3.4. *For each $k \geq 1$, $\mathrm{NPMV} \circ \mathrm{PF}^{\mathrm{NP}[k]} = \mathrm{NPMV} \circ \mathrm{PF}_{tt}^{\mathrm{NP}[2^k-1]}$.*

Proof. $\mathrm{NPMV} \circ \mathrm{PF}^{\mathrm{NP}[k]} \subseteq \mathrm{NPMV} \circ \mathrm{PF}_{tt}^{\mathrm{NP}[2^k-1]}$ follows immediately from $\mathrm{PF}^{\mathrm{NP}[k]} \subseteq \mathrm{PF}_{tt}^{\mathrm{NP}[2^k-1]}$ [2]. (Recall the comment that follows Proposition 2.3.)

Now we show $\mathrm{NPMV} \circ \mathrm{PF}_{tt}^{\mathrm{NP}[2^k-1]} \subseteq \mathrm{NPMV} \circ \mathrm{PF}^{\mathrm{NP}[k]}$. Let $f = g \circ h \in \mathrm{NPMV} \circ \mathrm{PF}_{tt}^{\mathrm{NP}[2^k-1]}$ with $g \in \mathrm{NPMV}$ and $h \in \mathrm{PF}_{tt}^{\mathrm{NP}[2^k-1]}$. Let $g \in \mathrm{NPMV}$ via a nondeterministic Turing machine N and let h' be a single-valued refinement of h that is computed by a deterministic oracle Turing machine transducer M with oracle $A \in \mathrm{NP}$. Define s to be a function that maps x to the number of queries in A that M makes on input x. s is a total, single-valued function. Define $B = \{\langle x, n \rangle \mid s(x) \geq n\}$. Clearly B belongs to NP. Since s is total and $0 \leq s(x) \leq 2^k - 1$, a binary search algorithm over $\{1, \cdots, 2^k - 1\}$ computes s in polynomial time with oracle B. The number of queries is exactly k, so $s \in \mathrm{PF}^{\mathrm{NP}[k]}$. Define s' to be a multivalued function

[2] Actually, it was shown there that $\mathrm{PF}^{\mathrm{NP}[\log]}/_{sv} \neq \mathrm{PF}^{\mathrm{NP}[\log]}/_{sv}$ unless $\mathrm{P} = \mathrm{FewP}$ and $\mathrm{R} = \mathrm{NP}$, but the two forms of the statement are equivalent by Proposition 2.3.

that maps x to $\langle x, n\rangle$ if and only if either

1. $n = s(x)$, or
2. $n = 2^k + w$ for some w such that $h(x)$ maps to w.

Clearly, s' has a refinement in $\mathrm{PF}^{\mathrm{NP}[k]}$, so $s' \in \mathrm{PF}^{\mathrm{NP}[k]}$ by Proposition 2.2.

Let $A \in \mathrm{NP}$ be witnessed by a machine D and define E to be the machine that on input $\langle x, n\rangle$ behaves as follows:

(1) If $n \geq 2^k$, then E sets $w = n - 2^k$ and goes to step (6).
(2) E computes the set Q of all query strings of M on x.
(3) E nondeterministically guesses $R \subseteq Q$ of size n.
(4) For each $y \in R$, E simulates D on y. If D on y does not accept for some $y \in R$, then E halts without accepting.
(5) E simulates M on x answering a query $q \in Q$ affirmatively if and only if $q \in R$. If M halts without accepting, so does E. Otherwise (M has an output), E computes the output w.
(6) E simulates N on w. If N outputs a string z, then so does E, and if it does not accept, then E halts without accepting.

Let $t \in \mathrm{NPMV}$ denote the partial multivalued function that E computes. We fix x and z in what follows. Suppose $f(x)$ maps to z. Then by definition there is a w such that $h(x) \mapsto w$ and $g(w) \mapsto z$, but in this case we know that $s'(x) \mapsto \langle x, 2^k + w\rangle$ and $t(x, 2^k + w)$ simulates N on w so $t(x, 2^k + w)$ maps to z. Thus $t \circ s'$ maps x to z.

Conversely, suppose $s'(x)$ maps to some $\langle x, n\rangle$ and $t(x, n)$ maps to z. If $n \geq 2^k$ then it must be that both $n = 2^k + w$ such that $h(x) \mapsto w$, and $g(w)$ maps to z ($t(x, n)$ just simulates N on $w = n - 2^k$). Thus $f(x)$ maps to z in this case. If $n \leq 2^k - 1$, then $n = s(x)$. We have $t(x, s(x)) \mapsto z$ if and only if there exist w and a set $R \subseteq A$ consisting of $s(x)$ query strings of M on x such that (i) given affirmative answers to all strings in R and negative answers to all strings in $Q - R$, M on x computes w and (ii) N on w outputs z on some computation path. Since $s(x)$ is exactly the number of query strings in A, $t(x, s(x)) = g(h'(x))$. Since $g \circ h'$ is a refinement of f, if $t(x, s(x))$ maps to z then $f(x)$ maps to z.

Therefore, $f = t \circ s' \in \mathrm{NPMV} \circ \mathrm{PF}^{\mathrm{NP}[k]}$. □

Remark. If we restrict s' in the above proof so that $s'(x)$ only maps to $\langle x, s(x)\rangle$, then s' is single valued and $t \circ s'$ is a refinement of f. This shows that $\mathrm{NPMV} \circ \mathrm{PF}_{tt}^{\mathrm{NP}[2^k-1]} \subseteq_c \mathrm{NPMV} \circ (\mathrm{PF}^{\mathrm{NP}[k]}/_{sv})$.

The left-to-right inclusion in the next theorem is completely analogous with the NP case, given by Beigel [2].

THEOREM 3.5. *For every $k \geq 1$, $\mathrm{PF}^{\mathrm{NPMV}[k]} = \mathrm{PF}_{tt}^{\mathrm{NPMV}[2^k-1]}$.*

Proof. Let $f \in \mathrm{PF}^{\mathrm{NPMV}[k]}$ be computed by a deterministic oracle Turing machine transducer M with $g \in \mathrm{NPMV}$ as the oracle. Let $h = s[M, g]$ in Proposition 3.1. Define D to be the machine with oracle h that, on input x, behaves as follows:

(1) D deterministically computes $w(0^k) = h(x, 0^k)$.
(2) For each $a \in \Sigma^k - \{0^k\}$, D sets $w(a)$ to the answer to $h(x, a)$. The number of queries to h is $2^k - 1$.
(3) D sets b to the largest a such that $w(a) \neq \perp$.
(4) If $w(b) = 0$, then D rejects x, and if $w(b) = 1y$ for some y, then D outputs y.

By Proposition 3.1, for every x, $D(x)$ rejects if and only if $x \notin \mathit{dom}(f)$, and if $D(x)$ outputs y, then $f(x) \mapsto y$. So, D computes a refinement of f. Thus, $f \in \mathrm{PF}_{tt}^{\mathrm{NPMV}[2^k-1]}$. This proves that $\mathrm{PF}^{\mathrm{NPMV}[k]} \subseteq \mathrm{PF}_{tt}^{\mathrm{NPMV}[2^k-1]}$

Now let $f \in \mathrm{PF}_{tt}^{\mathrm{NPMV}[2^k-1]}$ be computed by a deterministic oracle Turing machine transducer M with $g \in$ NPMV as the oracle. Let $h = s[M, g]$ in Proposition 3.2. Define D to be the machine with oracle h that, on input x, behaves as follows:

(1) D deterministically computes $w(0) = h(x, 0)$.

(2) By using a binary search over $[1, 2^k - 1]$ with oracle h, D computes $m = \max\{n \in \{0, \cdots, 2^k - 1\} \mid h(x, n) \text{ has an output }\}$. While doing this, D keeps the answers obtained from the oracle, and sets $w(m)$ to the answer to $h(x, m)$.

(3) If $w(m) = 0$, then D rejects x, and if $w(m) = 1y$ for some y, then D outputs y.

By Proposition 3.2, for every x, $D(x)$ rejects if and only if $x \notin dom(f)$ and, if $D(x)$ outputs y, then $f(x) \mapsto y$. So, D computes a refinement of f. Since the number of queries to h is k, we have $f \in \mathrm{PF}^{\mathrm{NPMV}[k]}$. Thus, $\mathrm{PF}_{tt}^{\mathrm{NPMV}[2^k-1]} \subseteq \mathrm{PF}^{\mathrm{NPMV}[k]}$, which proves the theorem. □

The above theorems yield the following corollary.

Corollary 3.6. *For every $k \geq 1$, $\mathrm{PF}^{\mathrm{NPMV}[k]} = \mathrm{PF}_{tt}^{\mathrm{NPMV}[2^k-1]} \subseteq_c \mathrm{NPMV} \circ (\mathrm{PF}^{\mathrm{NP}[k]}/_{sv}) \subseteq \mathrm{PF}^{\mathrm{NPMV}[k+1]} = \mathrm{PF}_{tt}^{\mathrm{NPMV}[2^{k+1}-1]}$.*

By Proposition 2.2, $\mathrm{PF}^{\mathrm{NPMV}[k]} \subseteq \mathrm{PF}^{\mathrm{NPMV}[k+1]}$. For general bounded query classes, it is not known whether $\mathrm{PF}^{\mathrm{NPMV}[k]} \subseteq \mathrm{PF}^{\mathrm{NP}[k]}$. But, for reduction classes of sets, this type of equivalence holds.

Theorem 3.7. *For every $k \geq 1$, $\mathrm{P}^{\mathrm{NPMV}[k]} = \mathrm{P}^{\mathrm{NP}[k]}$.*

Proof. Let $k \geq 1$. It suffices to show that $\mathrm{P}^{\mathrm{NPMV}[k]} \subseteq \mathrm{P}^{\mathrm{NP}[k]}$. Let $A \in \mathrm{P}^{\mathrm{NPMV}[k]}$. Then $f = \chi_A$ is in $\mathrm{PF}^{\mathrm{NPMV}[k]}$. Let M and $g \in$ NPMV be a machine and a partial multivalued function witnessing this property, respectively. Informally, we will show that $f \in \mathrm{P}^{\mathrm{NP}[k]}$ by using Proposition 3.1 to compute f. Namely, by letting $h = s[M, g]$ be the NPMV function given in Proposition 3.1, and letting a_x be the largest a in Σ^k such that $h(x, a)$ is defined, we will show that k queries to an NP oracle suffice both to find a_x and to compute $h(x, a_x) = 1f(x)$. Assume without loss of generality that M always outputs exactly one bit for all oracles. For simplicity, we fix x in the following discussion.

Let a_x be the largest $a \in \Sigma^k$ such that $h(x, a)$ is defined. The function $\lambda x.[h(x, a_x)]$ is total and single-valued, $x \in A$ if and only if $h(x, a_x) = 11$, and $x \notin A$ if and only if $h(x, a_x) = 10$. Let $b \in \Sigma$ such that $1b = h(x, 0^k)$ and ρ_x be the largest $r \in \Sigma^{k-1}$ such that either $h(x, r0)$ or $h(x, r1)$ maps to $1b$. It is not hard to see that $h(x, a_x) = 1b$ if and only if

(a) for every $a > \rho_x 1$, $h(x, a)$ maps to neither 10 nor 11, and

(b) $h(x, \rho_x 1)$ does not map to $1b'$, where $b' = 1$ if $b = 0$ and 0 otherwise.

Since $\rho_x \geq 0^{k-1}$ and $graph(h) \in$ NP, it is easy to see that ρ_x is computed by making $k-1$ questions to an NP oracle: we perform a binary search over Σ^{k-1} in order to find the largest $r \in \Sigma^{k-1}$ such that either $h(x, r0) \mapsto 1b$ or $h(x, r1) \mapsto 1b$. After ρ_x is found, conditions (a) and (b) can be tested by a single question to an NP oracle. Therefore, by making k queries to an NP oracle, $h(x, a_x)$ is computed. Since $h(x, a_x) = 11$ if and only if $x \in A$, this implies that $A \in \mathrm{P}^{\mathrm{NP}[k]}$. This proves the theorem. □

Note that Theorem 3.7 holds even if k is replaced by any polynomially bounded function. This means, remarkably, that in *any* polynomial time computation of a set relative to NPMV, the queries to NPMV can be replaced one for one with queries to NP.

Theorem 3.8. *For every $k \geq 1$, $\mathrm{P}_{tt}^{\mathrm{NPMV}[k]} = \mathrm{P}_{tt}^{\mathrm{NP}[k]}$.*

Proof. Let $k \geq 1$. It suffices to show that $\mathrm{P}_{tt}^{\mathrm{NPMV}[k]} \subseteq \mathrm{P}_{tt}^{\mathrm{NP}[k]}$. Let $A \in \mathrm{P}_{tt}^{\mathrm{NPMV}[k]}$. Then $f = \chi_A$ is in $\mathrm{PF}_{tt}^{\mathrm{NPMV}[k]}$. Let M and $g \in \mathrm{NPMV}$ be a machine and a partial multivalued function witnessing this property, respectively. We assume without loss of generality that M outputs 0 or 1 for all oracles and inputs. We will use Proposition 3.2 to compute f. Namely, we let $h = s[M, g]$ be the NPMV function given in Proposition 3.2, and fixing an input x, we let n_x be the largest n in $\{0, \cdots, k\}$ such that $h(x, n)$ is defined. We will show that k nonadaptive queries to an NP oracle suffice to compute $h(x, n_x) = 1f(x)$. The informal idea of the proof is as follows: we first compute b, the output of $M(x)$ where all query answers are $\perp$. Then we use nonadaptive NP queries to inspect sequences of query answers to $M(x)$ which are *plausible*, i.e., where all non-$\perp$ answers are verifiably legitimate outputs of the oracle function, but where each $\perp$ answer may or may not be correct. We look for m, the largest number of non-$\perp$ query answers in any plausible sequence of query answers where M outputs b. Actually, computing m exacty uses all our alloted NP queries without telling us the real output of M, so instead we only compute which of the pairs $\{2i, 2i + 1\}$ m belongs to. This only uses about $k/2$ nonadaptive queries. Simultaneously, for each pair $\{2i, 2i + 1\}$ we ask NP if *either* there is any plausible sequence with more than $2i+1$ non-$\perp$ answers, *or* M outputs $1-b$ for some plausible sequence containing exactly $2i + 1$ non-$\perp$ answers. The answer to this NP question for the pair containing m immediately tells us whether M outputs b or $1 - b$ for any correct sequence of query answers. This uses up the remaining $k/2$ NP queries.

We now present an exact version of the sketch above, showing that $f \in \mathrm{P}_{tt}^{\mathrm{NP}[k]}$. Note that the value of $h(x, 0)$ encodes the output of $M(x)$ with all queries answered with $\perp$, and $h(x, n)$ for $n > 0$ simply encodes the possible outputs of $M(x)$ over all plausible sequences of queries with at least n non-$\perp$ answers. Also, n_x is the exact number of non-$\perp$ entries in any sequence of correct query answers.) We are assuming M outputs exactly one bit for all possible sets of query answers, so h outputs nothing but 10 or 11.

By Proposition 3.2, $\lambda x.[h(x, n_x)]$ is total and single-valued, $x \in A$ if and only if $h(x, n_x) = 11$, and $x \notin A$ if and only if $h(x, n_x) = 10$. Let $b \in \Sigma$ be such that $1b = h(x, 0)$, and $d = \lfloor k/2 \rfloor$. Define two predicates S and T as follows:

$S(x, r)$ if and only if either $h(x, 2r)$ or $h(x, 2r + 1)$ maps to $1b$.

$T(x, r)$ if and only if

- $h(x, n)$ is defined for some $n > 2r + 1$, or
- $h(x, 2r + 1)$ maps to $1b'$, where $b' = 1 - b$.

Note that S and T are NP-predicates, $S(x, 0) = \mathit{true}$, and if k is even, then $T(x, d) = \mathit{false}$. Define ρ_x to be the largest $r \in \{0, \cdots, d\}$ such that $S(x, r) = \mathit{true}$. It is not hard to see that $h(x, n_x) = 1b$ if and only if $T(x, \rho_x) = \mathit{false}$.

Our goal is to compute $h(x, n_x)$ without making more than k queries to some NP oracle. Our method to accomplish this is to partition the domain $\{0, \cdots, k\}$ into successive pairs. For each pair $\{2r, 2r + 1\}$, we make two queries of the form "$S(x, r) = \mathit{true}$?" and "$T(x, r) = \mathit{true}$?". As observed above, $S(x, 0) = \mathit{true}$, and if k is even, $T(x, d) = \mathit{false}$. So, we need exactly k queries to S or T. Since S and T are NP-predicates, a single set in NP can answer both types of questions. Thus, by making k nonadaptive queries to an NP oracle, we determine whether $h(x, n_x) = 1b$ or not. Since $h(x, n_x) = 11$ if and only if $x \in A$, this implies that $A \in \mathrm{P}_{tt}^{\mathrm{NP}[k]}$. This proves the theorem. □

Our next goal is to show that bounded query hierarchies probably do not collapse.

LEMMA 3.9. *Let $k \geq 1$. If $\mathrm{PF}^{\mathrm{NPMV}[k+1]} = \mathrm{PF}^{\mathrm{NPMV}[k]}$, then for every $\ell \geq k$,*

$\mathrm{PF}^{\mathrm{NPMV}[\ell]} = \mathrm{PF}^{\mathrm{NPMV}[k]}$.

Proof. Let k be as in the hypothesis, let $m \geq 0$, and let $f \in \mathrm{PF}^{\mathrm{NPMV}[k+1+m]}$ be computed by a deterministic oracle Turing machine transducer M with $g \in \mathrm{NPMV}$ as the oracle. We will show that $f \in \mathrm{PF}^{\mathrm{NPMV}[k+m]}$.

Let N be an oracle transducer that on input x with oracle g outputs an ID of M on x just after obtaining the answer to its m-th query to g. Clearly, N makes at most m queries to g, and if g is single-valued, then $N[g]$ is total and single-valued. Define D to be the machine that, given an ID I of M, (1) attempts to simulate the computation of M starting from ID I by making at most $k+1$ queries to its oracle, and (2) if M halts without an output, then so does D and if M outputs a string y, then so does D.

It is not hard to see that D with oracle g defines a function r such that for every x and single-valued refinement g' of g, $f(x)$ is defined if and only if r maps the ID $N[g'](x)$ to some output y. And if the latter holds, then $f(x)$ also maps to y. Moreover, it is clear that $r \in \mathrm{PF}^{\mathrm{NPMV}[k+1]} = \mathrm{PF}^{\mathrm{NPMV}[k]}$, thus there is an oracle transducer Q such that for every single-valued refinement g' of g, $Q[g']$ computes a refinement of r making only k queries to g'. Now we combine the machines N and Q by taking the output of N as the input to Q. The resulting machine with oracle g' computes a refinement of f and makes at most $k+m$ queries to g'. Thus $f \in \mathrm{PF}^{\mathrm{NPMV}[k+m]}$ by Proposition 2.2.

By applying this argument repeatedly, we have $\mathrm{PF}^{\mathrm{NPMV}[\ell]} = \mathrm{PF}^{\mathrm{NPMV}[k]}$ for every $\ell \geq k$. $\square$

LEMMA 3.10. *Let $k \geq 1$. If $\mathrm{PF}_{tt}^{\mathrm{NPMV}[k+1]} = \mathrm{PF}_{tt}^{\mathrm{NPMV}[k]}$, then for every $\ell \geq k$, $\mathrm{PF}_{tt}^{\mathrm{NPMV}[\ell]} = \mathrm{PF}_{tt}^{\mathrm{NPMV}[k]}$.*

Proof. Let $k \geq 1$ and suppose that $\mathrm{PF}_{tt}^{\mathrm{NPMV}[k+1]} = \mathrm{PF}_{tt}^{\mathrm{NPMV}[k]}$. Let $m \geq 0$ and $f \in \mathrm{PF}_{tt}^{\mathrm{NPMV}[k+1+m]}$. We will show that $f \in \mathrm{PF}_{tt}^{\mathrm{NPMV}[k+m]}$. By applying the argument repeatedly, for every $\ell \geq k$, we have $\mathrm{PF}_{tt}^{\mathrm{NPMV}[\ell]} = \mathrm{PF}_{tt}^{\mathrm{NPMV}[k]}$.

Let $f \in \mathrm{PF}_{tt}^{\mathrm{NPMV}[k+1+m]}$ be computed by a deterministic oracle Turing machine transducer M with $g \in \mathrm{NPMV}$ as the oracle. For x, let $q_1(x), \cdots, q_{k+1+m}(x)$ denote the queries of M on x. Define c to be the function that maps x to $(y_1, \cdots, y_{k+1})$ so that for every $i, 1 \leq i \leq k+1$, y_i is a value of $g(q_i(x))$ if $q_i(x) \in \mathit{dom}(g)$ and $y_i = \perp$ otherwise. Obviously, $c \in \mathrm{PF}_{tt}^{\mathrm{NPMV}[k+1]}$, so by our supposition, $c \in \mathrm{PF}_{tt}^{\mathrm{NPMV}[k]}$. Let c be computed by a deterministic oracle Turing machine transducer N with $h \in \mathrm{NPMV}$ as the oracle. Then we can easily construct a machine that computes f by making k queries to h and m queries to g. Therefore, $f \in \mathrm{PF}_{tt}^{\mathrm{NPMV}[k+m]}$. This proves the lemma. $\square$

The Boolean hierarchy over NP is defined by Wagner and Wechsung [19] and has been studied extensively [6, 7, 8, 12]. We denote the k-th level of the Boolean hierarchy as $\mathrm{NP}(k)$. By definition,

1. $\mathrm{NP}(1) = \mathrm{NP}$, and
2. for every $k \geq 2$, $\mathrm{NP}(k) = \mathrm{NP} - \mathrm{NP}(k-1)$.

The Boolean hierarchy over NP, denoted by BH is the union of all $\mathrm{NP}(k)$, $k \geq 1$.

Kadin [12] proved that the Boolean hierarchy collapses only if the polynomial-time hierarchy collapses.

THEOREM 3.11. *Let $k \geq 1$. If $\mathrm{PF}^{\mathrm{NPMV}[k+1]} = \mathrm{PF}^{\mathrm{NPMV}[k]}$, then* BH *collapses to its 2^k-th level.*

Proof. Suppose that $\mathrm{PF}^{\mathrm{NPMV}[k+1]} = \mathrm{PF}^{\mathrm{NPMV}[k]}$. By Lemma 3.9 and Theorem 3.5, for every $m > k$, $\mathrm{PF}^{\mathrm{NPMV}[m]} \subseteq \mathrm{PF}^{\mathrm{NPMV}[k]} = \mathrm{PF}_{tt}^{\mathrm{NPMV}[2^k-1]}$. So, by Theorem 3.7 and results by Köbler, Schöning, and Wagner [14], we have, for every $m > k$, $\mathrm{P}^{\mathrm{NP}[m]} = \mathrm{P}_{tt}^{\mathrm{NP}[2^k-1]} \subseteq \mathrm{NP}(2^k)$. Thus, $\mathrm{BH} = \mathrm{NP}(2^k)$. □

The following theorem is proved in a similar manner.

THEOREM 3.12. *Let* $k \geq 1$. *If* $\mathrm{PF}_{tt}^{\mathrm{NPMV}[k+1]} = \mathrm{PF}_{tt}^{\mathrm{NPMV}[k]}$, *then* BH *collapses to its* $(k+1)$*-st level.*

Proof. Suppose that $\mathrm{PF}_{tt}^{\mathrm{NPMV}[k+1]} = \mathrm{PF}_{tt}^{\mathrm{NPMV}[k]}$. By Lemma 3.10, for every $m > k$, $\mathrm{PF}_{tt}^{\mathrm{NPMV}[m]} \subseteq \mathrm{PF}_{tt}^{\mathrm{NPMV}[k]}$. So, by Theorem 3.8 and results by Köbler, Schöning, and Wagner [14], we have, for every $m > k$, $\mathrm{P}_{tt}^{\mathrm{NP}[m]} = \mathrm{P}_{tt}^{\mathrm{NP}[k]} \subseteq \mathrm{NP}(k+1)$. Thus, $\mathrm{BH} = \mathrm{NP}(k+1)$. □

Analogous to the theorems stated so far, the following theorems hold for reduction classes that make logarithmic many queries to partial functions in NPMV. We see in these theorems a different behavior when computing partial multivalued functions with bounded queries to NPMV than when computing partial functions with bounded queries to NP. To wit, in contrast to the following results, Selman [15] shows that $\mathrm{PF}^{\mathrm{NP}[\log]} = \mathrm{PF}_{tt}^{\mathrm{NP}}$ only if P = FewP and R = NP. The reason seems to be, as we showed in Theorems 3.3 and 3.5, that the mind-change argument [2, 19] works for $\mathrm{PF}^{\mathrm{NPMV}}$ (as it does for P^{NP}), but apparently does not work for $\mathrm{PF}^{\mathrm{NP}}$.

THEOREM 3.13.

1. $\mathrm{PF}^{\mathrm{NPMV}[\log]} = \mathrm{PF}_{tt}^{\mathrm{NPMV}} \subseteq \mathrm{NPMV} \circ \mathrm{PF}^{\mathrm{NP}[\log]} = \mathrm{NPMV} \circ \mathrm{PF}_{tt}^{\mathrm{NP}}$.
2. $\mathrm{NPMV} \circ (\mathrm{PF}^{\mathrm{NP}[\log]}/_{sv}) \subseteq \mathrm{PF}^{\mathrm{NPMV}[\log]}$.
3. $\mathrm{PF}^{\mathrm{NPMV}[\log]} \subseteq_c \mathrm{NPMV} \circ (\mathrm{PF}^{\mathrm{NP}[\log]}/_{sv})$.
4. $\mathrm{NPMV} \circ (\mathrm{PF}^{\mathrm{NP}[\log]}/_{sv}) \subseteq \mathrm{NPMV} \circ (\mathrm{PF}_{tt}^{\mathrm{NP}}/_{sv})$.
5. $\mathrm{NPMV} \circ (\mathrm{PF}_{tt}^{\mathrm{NP}}/_{sv}) \subseteq_c \mathrm{NPMV} \circ (\mathrm{PF}^{\mathrm{NP}[\log]}/_{sv})$.

Proof. Note for any function t such that $t(x) \leq c \log |x|$, that $2^{t(x)} - 1$ is polynomially bounded, and, for any polynomial p, that $\log p(|x|) \leq c \log |x|$ for some constant c. Therefore, a proof similar to that of Theorem 3.5 shows that $\mathrm{PF}^{\mathrm{NPMV}[\log]} = \mathrm{PF}_{tt}^{\mathrm{NPMV}}$, and a proof similar to Theorem 3.3 (2) shows that $\mathrm{PF}^{\mathrm{NPMV}[\log]} \subseteq \mathrm{NPMV} \circ \mathrm{PF}^{\mathrm{NP}[\log]}$. A proof similar to that of Theorem 3.4 yields $\mathrm{NPMV} \circ \mathrm{PF}^{\mathrm{NP}[\log]} = \mathrm{NPMV} \circ \mathrm{PF}_{tt}^{\mathrm{NP}}$. Thus (1) holds.

Inclusion (2) follows by a straightforward simulation, and (3) follows by adapting the proof of Theorem 3.3 (1). Using a technique of Buss and Hay [4], for any set A, logarithmically many adaptive queries to A can be simulated by polynomially many nonadaptive queries to A, so $\mathrm{PF}^{\mathrm{NP}[\log]}/_{sv} \subseteq \mathrm{PF}_{tt}^{\mathrm{NP}}/_{sv}$. Thus, $\mathrm{NPMV} \circ (\mathrm{PF}^{\mathrm{NP}[\log]}/_{sv}) \subseteq \mathrm{NPMV} \circ (\mathrm{PF}_{tt}^{\mathrm{NP}}/_{sv})$. Hence (4) holds. Inclusion (5) follows by adapting the remark following the proof of Theorem 3.4. □

THEOREM 3.14. $\mathrm{P}^{\mathrm{NPMV}[\log]} = \mathrm{P}_{tt}^{\mathrm{NPMV}} = \mathrm{P}^{\mathrm{NP}[\log]} = \mathrm{P}_{tt}^{\mathrm{NP}}$.

The proof is similar to those of Theorem 3.7 and Theorem 3.8.

4. Nondeterministic Polynomial-Time Reductions.

We define nondeterministic reductions between partial functions so that the access mechanism is identical to that for deterministic reductions. Namely, let g be a single-valued partial function and N be a polynomial time nondeterministic oracle Turing machine. $N[g]$ denotes a multivalued partial function computed by N with oracle g in accordance with the following mechanism:

- when N asks about $y \in dom(g)$, g returns $g(y)$ and
- when N asks about $y \notin dom(g)$, g answers a special symbol $\perp$.

DEFINITION 4.1. *Let f and g be partial multivalued functions. We say that f* is nondeterministic polynomial-time Turing reducible to g, *denoted by* $f \leq_{\mathrm{T}}^{\mathrm{NP}} g$, *if* *there is a polynomial time nondeterministic Turing machine N such that for every single-valued refinement g' of g, $N[g']$ is a refinement of f.*

Thus, for every x and for every single-valued refinement g' of g,

- $x \in dom(f)$ if and only if $x \in dom(N[g'])$ and
- if $N[g']$ maps x to y, then f maps x to y.

Let $\mathcal{F}$ be a class of partial multivalued functions. $\mathrm{NPMV}^{\mathcal{F}}$ denotes the class of partial multivalued functions that are $\leq_{\mathrm{T}}^{\mathrm{NP}}$-reducible to some $g \in \mathcal{F}$. $\mathrm{NPMV}^{\mathcal{F}[k]}$ denotes the class of partial multivalued functions that are $\leq_{\mathrm{T}}^{\mathrm{NP}}$-reducible to some $g \in \mathcal{F}$ via a machine that makes k adaptive queries to its oracle.

For a class of sets $\mathcal{C}$, we write $\mathrm{NPMV}^{\mathcal{C}}$ to denote the class of multivalued partial functions that are computed by a nondeterministic Turing machine relative to an oracle in $\mathcal{C}$. $\mathrm{NPMV}^{\mathcal{C}[k]}$ is defined similarly.

It is easy to see that every nondeterministic polynomial time reduction to partial functions is replaceable by a reduction that makes nonadaptive queries to its oracle and that preserves the number of queries. For this reason, we do not distinguish classes $\mathrm{NPMV}_{tt}^{\mathcal{F}}$ or $\mathrm{NPMV}_{tt}^{\mathcal{F}[k]}$.

For $k \geq 1$, $\Sigma\mathrm{MV}_k$ denotes $\underbrace{\mathrm{NPMV}^{\cdot^{\cdot^{\mathrm{NPMV}}}}}_{k}$.

LEMMA 4.1. *For every $k \geq 1$, $\Sigma\mathrm{MV}_k = \mathrm{NPMV}^{\Sigma_{k-1}^{\mathrm{P}}[1]}$ and for every $f \in \Sigma\mathrm{MV}_k$, $dom(f) \in \Sigma_k^{\mathrm{P}}$.*

Proof. The proof is by an induction on k. The statement trivially holds for $k = 1$. Let $k = 2$. We show that $\mathrm{NPMV}^{\mathrm{NPMV}} \subseteq \mathrm{NPMV}^{\mathrm{NP}[1]}$. Let $f \in \mathrm{NPMV}^{\mathrm{NPMV}}$ via a machine M and a function $g \in \mathrm{NPMV}$. Let N be a machine witnessing $g \in \mathrm{NPMV}$. Define A to be the set of all $(y_1, \cdots, y_m)$, $m \geq 1$, such that $y_1, \cdots, y_m \notin dom(g)$. Obviously, $A \in$ co-NP. Consider a nondeterministic Turing machine T that, on input x, simulates M on x in the following way:

- When M queries about string w, T simulates N on w. If N on w outputs a string z, then T assumes that the answer from the oracle is z and if N on w does not accept, then T assumes that the answer from the oracle is $\perp$.
- When M enters a halting state, T enumerates all the queries w for which the answer from the oracle are assumed to be $\perp$. T sets $y_1, \cdots, y_m$ to the enumeration.
  - If $(y_1, \cdots, y_m) \notin A$, then T rejects,
  - if $(y_1, \cdots, y_m) \in A$ and M rejects, then T rejects, and
  - if $(y_1, \cdots, y_m) \in A$ and M outputs some string z, then T outputs z.

It is not hard to see that for every x and z, $f(x)$ maps to z if and only if there is a computation of M relative to g that outputs z if and only if there is a computation of T relative to A that outputs z. Thus T relative to A computes a refinement of f. Furthermore, T is polynomial time-bounded and T makes only one question to A. So, $f \in \mathrm{NPMV}^{\mathrm{NP}[1]}$. Thus, $\mathrm{NPMV}^{\mathrm{NPMV}} = \mathrm{NPMV}^{\mathrm{NP}[1]}$. Moreover, by the above discussions, for every function $f \in \mathrm{NPMV}^{\mathrm{NPMV}}$, $dom(f) \in \Sigma_2^{\mathrm{P}}$.

Now let $k \geq 2$ and suppose that the claim holds for every $k' \leq k$. Since the above proof is relativizable, for any class $\mathcal{C}$ of sets, we have $\mathrm{NPMV}^{\mathrm{NPMV}^{\mathcal{C}}} \subseteq \mathrm{NPMV}^{\mathrm{NP}^{\mathcal{C}}[1]}$. So, $\mathrm{NPMV}^{\Sigma\mathrm{MV}_k} = \mathrm{NPMV}^{\mathrm{NPMV}^{\Sigma_{k-1}^{\mathrm{P}}[1]}}$ (by induction hypothesis) $\subseteq \mathrm{NPMV}^{\mathrm{NPMV}^{\Sigma_{k-1}^{\mathrm{P}}}} \subseteq \mathrm{NPMV}^{\mathrm{NP}^{\Sigma_{k-1}^{\mathrm{P}}}[1]} = \mathrm{NPMV}^{\Sigma_k^{\mathrm{P}}[1]}$. This proves the lemma. □

This lemma yields the following theorem.

THEOREM 4.2. *Let f be a partial multivalued function. For every $k \geq 1$, the following statements are equivalent:*

(i) *f is in $\Sigma\mathrm{MV}_k$;*

(ii) *f has a polynomially length-bounded refinement g such that $dom(g) \in \Sigma_k^{\mathrm{P}}$, and $graph(g) \in \Sigma_k^{\mathrm{P}}$;*

(iii) *f has a polynomial length-bounded refinement g such that $graph(g) \in \Sigma_k^{\mathrm{P}}$.*

THEOREM 4.3. *For every $k \geq 1$, $\Sigma\mathrm{MV}_{k+1} = \Sigma\mathrm{MV}_k$ if and only if $\Sigma_{k+1}^{\mathrm{P}} = \Sigma_k^{\mathrm{P}}$.*

Proof. First suppose that $\Sigma_{k+1}^{\mathrm{P}} = \Sigma_k^{\mathrm{P}}$. Let f be any function in $\Sigma\mathrm{MV}_{k+1}$. By Theorem 4.2 (ii), there is a polynomially length-bounded refinement g of f such that $dom(g), graph(g) \in \Sigma_{k+1}^{\mathrm{P}}$, so by our supposition, $dom(g), graph(g) \in \Sigma_k^{\mathrm{P}}$. Therefore, f is in $\Sigma\mathrm{MV}_k$. Hence, $\Sigma\mathrm{MV}_{k+1} = \Sigma\mathrm{MV}_k$.

Next suppose that $\Sigma\mathrm{MV}_{k+1} = \Sigma\mathrm{MV}_k$. Let A be $\leq_m^{\mathrm{P}}$-complete for $\Sigma_{k+1}^{\mathrm{P}}$. Define χ_A^0 to be the function that $\chi_A^0(x) = 1$ if $x \in A$ and undefined otherwise. Obviously, χ_A^0 is in $\Sigma\mathrm{MV}_{k+1}$, so by our supposition, $\chi_A^0 \in \Sigma\mathrm{MV}_k$. On the other hand, $dom(\chi_A^0) = A$. Thus, we have $A \in \Sigma_k^{\mathrm{P}}$. Since A is complete for $\Sigma_{k+1}^{\mathrm{P}}$, we have $\Sigma_{k+1}^{\mathrm{P}} = \Sigma_k^{\mathrm{P}}$. □

Thus, these classes form function analogues of the polynomial hierarchy, and, unless the polynomial hierarchy collapses, they form a proper hierarchy.

5. The Difference Hierarchy. Let $\mathcal{F}$ be a class of partial multivalued functions. A partial multivalued function f is in co$\mathcal{F}$ if there exist $g \in \mathcal{F}$ and a polynomial p such that for every x and y

- $f(x)$ maps to y if and only if $|y| \leq p(|x|)$ and $g(x)$ does not map to y.

Let $\mathcal{F}$ and $\mathcal{G}$ be two classes of partial multivalued functions. A partial multivalued function h is in $\mathcal{F} \wedge \mathcal{G}$ if there exist partial multivalued functions $f \in \mathcal{F}$ and $g \in \mathcal{G}$ such that for every x and y,

- $h(x)$ maps to y if and only if $f(x)$ maps to y and $g(x)$ maps to y.

A partial multivalued function h is in $\mathcal{F} \vee \mathcal{G}$ if there exist partial multivalued functions $f \in \mathcal{F}$ and $g \in \mathcal{G}$ such that for every x and y,

- $h(x)$ maps to y if and only if $f(x)$ maps to y or $g(x)$ maps to y.

$\mathcal{F} - \mathcal{G}$ denotes $\mathcal{F} \wedge$ co$\mathcal{G}$.

NPMV(k) is the class of partial multivalued functions defined in the following way:

1. NPMV(1) = NPMV, and
2. for $k \geq 2$, NPMV(k) = NPMV − NPMV($k-1$).

LEMMA 5.1. *For every $k \geq 1$, $f \in$ NPMV(k) if and only if f is polynomially length-bounded and $graph(f) \in$ NP(k).*

Proof. The proof is by an induction on k. For $k = 1$, the claim trivially holds. Let $k \geq 2$ and suppose that the claim holds for all $k' < k$. Let $f \in$ NPMV(k). There are functions $g \in$ NPMV and $h \in$ NPMV($k-1$) such that for every x and y, f maps x to y if and only if g maps x to y but h does not map x to y. So, $graph(f) = graph(g) - graph(h)$. By our hypothesis, $graph(h)$ belongs to NP($k-1$). Since $graph(g) \in$ NP, we have $graph(f) =$ NP(k).

On the other hand, let f be a polynomially length bounded function whose graph is in NP(k). There are sets $A \in$ NP and $B \in$ NP($k-1$) such that $graph(f) = A - B$. Define g and h to be functions such that $graph(g) = A$ and $graph(h) = B$. By our hypothesis, $g \in$ NPMV and $h \in$ NPMV($k-1$). Therefore, $f \in$ NPMV(k). This proves the lemma. □

We use the above lemma to obtain the following theorem.

THEOREM 5.2. *For every $k \geq 1$,* NPMV($k+1$) = NPMV(k) *if and only if* NP($k+1$) = NP(k).

Despite the similarity in appearance, the difference hierarchy over NPMV is probably much stronger than both the difference hierarchy over NP and the bounded query hierarchy over NPMV. For example, it is well-known that *maxsat* is complete for $\mathrm{PF}^{\mathrm{NP}} = \mathrm{PF}^{\mathrm{NPMV}}$ [13]. Nonetheless, we have the following:

PROPOSITION 5.3. *maxsat* $\in$ NPMV(2).

Proof. Let $f \in$ NPMV be the function that maps x to y if and only if there is a $z > y$ such that z is a satisfying assignment for x. Clearly, for all x, $maxsat(x) = y$ if and only if $sat(x)$ maps to y and $f(x)$ does not map to y. Therefore, *maxsat* $\in$ NPMV(2). □

PROPOSITION 5.4. co(coNPMV) = NPMV.

Proof. Let $f \in$ NPMV. Let p be a polynomial such that for every x and y, if $f(x)$ maps to y, then $|y| \leq p(|x|)$. Let g be the complement of f with respect to p such that for every x and y, $g(x)$ maps to y if and only if $f(x)$ does not map to y and $|y| \leq p(|x|)$. Furthermore, let h be the complement of g with respect to p such that for every x and y, $h(x)$ maps to y if and only if $g(x)$ does not map to y and $|y| \leq p(|x|)$. For every x and y, $h(x)$ maps to y if and only if $f(x)$ maps to y. This implies $h = f$. Therefore, NPMV $\subseteq$ co(coNPMV).

Conversely, let $f \in$ co(coNPMV). There exist $g \in$ coNPMV and a polynomial p such that for every x and y, $f(x)$ maps to y if and only if $g(x)$ does not map to y and $|y| \leq p(|x|)$. Moreover, since $g \in$ coNPMV, there exist $h \in$ NPMV and a polynomial q such that for every x and y, $g(x)$ maps to y if and only if $h(x)$ does not map to y and $|y| \leq q(|x|)$. For every x and y, $f(x)$ maps to y if and only if either ($h(x)$ maps to y and $|y| \leq p(|x|)$) or ($q(|x|) < |y| \leq p(|x|)$). Let M be a nondeterministic Turing machine that computes h. Define N to be the machine that, on input x, (1) nondeterministically guesses $b \in \{0, 1\}$, (2) if $b = 0$, then simulates M on input x and outputs y if M outputs y and $|y| \leq p(|x|)$, and (3) if $b = 1$, then nondeterministically guesses $y, q(|x|) < |y| \leq p(|x|)$ and outputs y. Clearly, N computes f. So, co(coNPMV) $\subseteq$ NPMV. □

THEOREM 5.5. *The following statements are all equivalent.*

(a) NP = co-NP.
(b) NPMV $\subseteq$ coNPMV.
(c) coNPMV $\subseteq$ NPMV.

Proof. By Proposition 5.4, (b) is equivalent to (c). So, it suffices to show that (a) is equivalent to (c). First suppose that coNPMV $\subseteq$ NPMV. Define f to be the function that maps x to each of the three strings $\lambda, 0$, and 1 if $x \in$ SAT and undefined otherwise. Obviously, $f \in$ NPMV. Let p be a polynomial such that $p(n) = 1$ for all n. By taking the complement of f with respect to p, we obtain a function $g \in$ coNPMV that maps x to $\lambda, 0, 1$ if $x \notin$ SAT and undefined otherwise. So, $dom(g) = \overline{\mathrm{SAT}}$. Now by our supposition, we have $g \in$ NPMV, so $dom(g) \in$ NP. This implies $\overline{\mathrm{SAT}} \in$ NP, and thus, NP = co-NP.

Conversely, suppose that NP = co-NP. Let $f \in$ coNPMV. There exist $g \in$ NPMV and a polynomial p such that for every x and y, $f(x)$ maps to y if and only if $g(x)$ does not map to y and $|y| \leq p(|x|)$. The set of all (x, y) such that $g(x)$ does not map to y and $|y| \leq p(|x|)$ is in co-NP, so, by our supposition, it is in NP. Thus, $graph(f) \in$ NP, so $f \in$ NPMV. Hence, coNPMV $\subseteq$ NPMV. □

Define a function f to be NPMV-*low* if $\mathrm{NPMV}^f \subseteq_c$ NPMV.

THEOREM 5.6. *A function f is* NPMV-*low if and only if $f \in_c$* NPMV *with $dom(f) \in$* NP $\cap$ co-NP.

Proof. Let f be NPMV-low. Since $f \in \mathrm{NPMV}^f$ and $\mathrm{NPMV}^f \subseteq_c \mathrm{NPMV}$, $f \in_c$ NPMV. So, $dom(f) \in \mathrm{NP}$. Let $A = \overline{dom(f)}$. We wish to show that A belongs to NP. Define M to be a machine that, on input x, queries $f(x)$ and outputs 1 if $f(x) = \perp$ and 0 otherwise. The function h that M computes is the characteristic function of A. Since f is NPMV-low and h is single-valued, $h \in \mathrm{NPMV}$, so $A \in \mathrm{NP}$.

Conversely, let $f \in_c$ NPMV with $dom(f) \in \mathrm{NP} \bigcap \text{co-NP}$. Define A to be the set of all $(y_1, \cdots, y_m), m \geq 1$, such that $y_1, \cdots, y_m \notin dom(f)$. By our supposition, $A \in \mathrm{NP} \bigcap \text{co-NP}$. Let $g \in \mathrm{NPMV}^f$ via a machine M. By the proof of Lemma 4.1, there is a polynomial time nondeterministic Turing machine N that computes a refinement of g by making at most one query to A per computation path. Since $A \in \mathrm{NP} \bigcap \text{co-NP}$, the query to A can be simulated nondeterministically. So, $g \in_c$ NPMV. Therefore, f is NPMV-low. □

PROPOSITION 5.7. *For every $k \geq 1$ and $f \in \mathrm{NPMV}(k)$, $dom(f) \in \Sigma_2^{\mathrm{P}}$.*

Proof. By Lemma 5.1, f is polynomially length-bounded and $graph(f) \in \mathrm{NP}(k) \subseteq \Sigma_2^{\mathrm{P}}$. There is a polynomial p such that for all x,

$$x \in dom(f) \leftrightarrow (\exists y)[|y| \leq p(|x|) \wedge y \in graph(f)].$$

Thus $dom(f) \in \Sigma_2^{\mathrm{P}}$. □

We show next that Proposition 5.7 is tight, even when $k = 2$.

PROPOSITION 5.8. *Let A be in Σ_2^{P} via a polynomial p and a set B in* co-NP *so that for every x,*

$$x \in A \leftrightarrow (\exists y : |y| \leq p(|x|))[(x, y) \in B].$$

Let f be a function such that for every x and y,

$$f(x) \mapsto y \leftrightarrow |y| \leq p(|x|) \wedge (x, y) \in B.$$

Then, $f \in \mathrm{NPMV}(2)$. (Note that $dom(f) = A$.)

Proof. Let A, p, B, and f be as in the hypothesis. Define f_1 to be a function that maps x to each string y in $\Sigma^{\leq p(|x|)}$, and define f_2 to be a function that maps x to each string y in $\Sigma^{\leq p(|x|)}$ such that $(x, y) \notin B$. Obviously, $f_1, f_2 \in \mathrm{NPMV}$. For every x and y, $f(x)$ maps to y if and only if $f_1(x)$ maps to y and $f_2(x)$ does not map to y. So, $f \in \mathrm{NPMV}(2)$. □

By Theorem 5.2, the difference hierarchy for partial multivalued functions rises or falls in accordance with the difference hierarchy for sets. Since the difference hierarchy for sets sits entirely within Δ_2^{P}, one might anticipate that the NPMV(k) hierarchy lies within the second level of the polynomial-time hierarchy for partial multivalued functions. The following striking theorem shows that this can be true if and only if the polynomial-time hierarchy collapses.

THEOREM 5.9. $\mathrm{NPMV}(2) \subseteq \mathrm{PF}^{\mathrm{NPMV}}$ *if and only if* $\Sigma_2^{\mathrm{P}} = \Delta_2^{\mathrm{P}}$.

Proof. First suppose that $\mathrm{NPMV}(2) \subseteq_c \mathrm{PF}^{\mathrm{NPMV}}$. Let A be in Σ_2^{P}. By the above proposition, there is a function $f \in \mathrm{NPMV}(2)$ such that $dom(f) = A$. By our supposition, $f \in_c \mathrm{PF}^{\mathrm{NPMV}}$. So, there exist a polynomial time deterministic Turing machine M and a function $g \in \mathrm{NPMV}$ such that for every x, $x \in A$ if and only if $M(x)$ relative to g has an output. By modifying M slightly, we have a machine M' such that for every x, $M'(x)$ relative to g outputs 1 if $x \in A$ and 0 otherwise. Thus, $A \in \mathrm{P}^{\mathrm{NPMV}}$, and thus, $A \in \mathrm{P}^{\mathrm{NP}}$. Hence, $\Sigma_2^{\mathrm{P}} = \Delta_2^{\mathrm{P}}$.

Next suppose that $\Sigma_2^{\mathrm{P}} = \Delta_2^{\mathrm{P}}$. Let $f \in \mathrm{NPMV}(2)$. By Lemma 5.1, there exist $f_1, f_2 \in \mathrm{NPMV}$ such that $graph(f) = graph(f_1) - graph(f_2)$. Define A to be the set of all pairs (x, y) for which there is some $z \geq y$ such that $(x, z) \in \mathit{graph}(f_1) - \mathit{graph}(f_2)$. Define g to be the partial function that maps x to the largest y such that $(x, y) \in A$ if $x \in \mathit{dom}(f)$. It is not hard to see that g is a single-valued refinement of f and g is polynomial time computable with oracle A by an obvious binary search algorithm. By definition, $A \in \Sigma_2^{\mathrm{P}}$. So, $A \in \Delta_2^{\mathrm{P}}$. Therefore, $g \in \mathrm{PF}^{\Delta_2^{\mathrm{P}}} = \mathrm{PF}^{\mathrm{P}^{\mathrm{NP}}} = \mathrm{PF}^{\mathrm{NP}}$. Thus, $f \in_c \mathrm{PF}^{\mathrm{NPMV}}$. □

Theorem 5.9 raises a question, "how powerful is the difference hierarchy." The following results provide some answers to the question.

LEMMA 5.10. $\mathrm{PF}_{tt}^{\mathrm{NPMV}[k]} \subseteq_c \mathrm{NPMV}(2k+1)$.

Proof. Let $f \in \mathrm{PF}_{tt}^{\mathrm{NPMV}[k]}$ via a polynomial time deterministic Turing machine M that makes nonadaptive queries to a function $g \in \mathrm{NPMV}$. Let p be a polynomial such that for every x, each query string of M on x is of length $\leq p(|x|)$. Let $h = s[M, g]$ defined in Proposition 3.2. For each x, let n_x be the largest $n \in \{0, \cdots, k\}$ such that $h(x, n)$ is defined. Then for every x,

- either $h(x, n_x)$ maps only to 0 or $h(x, n_x)$ maps only to strings of the form $1z$, and
- if $h(x, n_x)$ maps to 0, then $x \notin \mathit{dom}(f)$ and if $h(x, n_x)$ maps to some $1z$, then f maps x to z.

For each $i \in \{0, \cdots, k\}$, define H_i to be the function that maps x to y if and only if $h(x, i)$ maps to $1y$. For each $i \in \{1, \cdots, k\}$, define G_i to be the function that maps x to each string in $\Sigma^{\leq p(|x|)}$ if for some $j > i$, $(x, j) \in \mathit{dom}(h)$ and undefined otherwise. These functions are obviously in NPMV. Note the following:

(1) for every $i > n_x$, $H_i(x)$ is undefined;
(2) for every $i \geq n_x$, $G_i(x)$ is undefined; and,
(3) for every $i < n_x$, $G_i(x)$ maps to every $y \in \Sigma^{\leq p(|x|)}$.

Define F by

$$\mathit{graph}(F) = \mathit{graph}(H_k) \cup \left(\bigcup\nolimits_{0 \leq i \leq k-1} (\mathit{graph}(H_i) - \mathit{graph}(G_i)) \right).$$

Then $\mathit{dom}(f) = \mathit{dom}(F)$ and if $F(x)$ maps to y then $f(x)$ maps to y. Therefore, F is a refinement of f. Since H_i and G_i are in NPMV,

$$\mathit{graph}(F) \in \mathrm{NP} \vee \underbrace{\mathrm{NP}(2) \vee \cdots \vee \mathrm{NP}(2)}_{k}.$$

So, $\mathit{graph}(F) \in \mathrm{NP}(2k+1)$. Therefore, $f \in_c \mathrm{NPMV}(2k+1)$. □

Since $\mathrm{PF}_{tt}^{\mathrm{NPMV}[2^k-1]} = \mathrm{PF}^{\mathrm{NPMV}[k]}$, we have the following theorem.

THEOREM 5.11. $\mathrm{PF}^{\mathrm{NPMV}[k]} \subseteq_c \mathrm{NPMV}(2^{k+1}-1)$.

By Theorem 5.2, the levels of the difference hierarchy of partial functions are distinct if and only if the same levels of the Boolean hierarchy are distinct. Yet, whereas the Boolean hierarchy resides entirely within P^{NP}, by Theorem 5.9, this is unlikely to be true of the difference hierarchy of partial functions.

6. Reduction classes to NPSV.

In this section, we set down some results about reduction classes to NPSV. With two notable exceptions, all of our results are corollaries of theorems that we already proved, and our interest is primarily in the following Corollaries 6.4 and 6.5, which demonstrate that bounded query hierarchies with oracles in NPSV do not collapse unless the Boolean hierarchy collapses.

The following proposition is easy to prove.

PROPOSITION 6.1.

1. $\mathrm{PF}^{\mathrm{NP}} = \mathrm{PF}^{\mathrm{NPSV}}$. *(by Theorem 2.4)*
2. $\mathrm{PF}^{\mathrm{NP}[k]} \subseteq \mathrm{PF}^{\mathrm{NPSV}[k]} \subseteq \mathrm{PF}^{\mathrm{NPMV}[k]}$ *and* $\mathrm{PF}^{\mathrm{NP}[\log]} \subseteq \mathrm{PF}^{\mathrm{NPSV}[\log]} \subseteq \mathrm{PF}^{\mathrm{NPMV}[\log]}$.
3. $\mathrm{PF}_{tt}^{\mathrm{NP}[k]} \subseteq \mathrm{PF}_{tt}^{\mathrm{NPSV}[k]} \subseteq \mathrm{PF}_{tt}^{\mathrm{NPMV}[k]}$.
4. $\mathrm{P}_{tt}^{\mathrm{NP}} = \mathrm{P}_{tt}^{\mathrm{NPSV}}$. *(by Theorem 3.14)*
5. $\mathrm{P}^{\mathrm{NP}} = \mathrm{P}^{\mathrm{NPSV}}$. *(by Theorem 2.4)*
6. $\mathrm{P}^{\mathrm{NP}[k]} \subseteq \mathrm{P}^{\mathrm{NPSV}[k]} \subseteq \mathrm{P}^{\mathrm{NPMV}[k]}$ *and* $\mathrm{P}^{\mathrm{NP}[\log]} \subseteq \mathrm{P}^{\mathrm{NPSV}[\log]} \subseteq \mathrm{P}^{\mathrm{NPMV}[\log]}$.
7. $\mathrm{P}_{tt}^{\mathrm{NP}[k]} \subseteq \mathrm{P}_{tt}^{\mathrm{NPSV}[k]} \subseteq \mathrm{P}_{tt}^{\mathrm{NPMV}[k]}$.

COROLLARY 6.2. *For every $k \geq 1$,* $\mathrm{P}^{\mathrm{NPSV}[k]} = \mathrm{P}^{\mathrm{NP}[k]}$.

Proof. By Proposition 6.1 (6), $\mathrm{P}^{\mathrm{NP}[k]} \subseteq \mathrm{P}^{\mathrm{NPSV}[k]} \subseteq \mathrm{P}^{\mathrm{NPMV}[k]}$. By Theorem 3.7, $\mathrm{P}^{\mathrm{NPMV}[k]} = \mathrm{P}^{\mathrm{NP}[k]}$. So, $\mathrm{P}^{\mathrm{NPSV}[k]} = \mathrm{P}^{\mathrm{NP}[k]}$. □

COROLLARY 6.3. *For every $k \geq 1$,* $\mathrm{P}_{tt}^{\mathrm{NPSV}[k]} = \mathrm{P}_{tt}^{\mathrm{NP}[k]}$.

COROLLARY 6.4. *If* $\mathrm{PF}_{tt}^{\mathrm{NPSV}[k+1]} = \mathrm{PF}_{tt}^{\mathrm{NPSV}[k]}$ *for some $k \geq 1$, then* BH *collapses to its $(k+1)$-st level.*

Proof. Suppose $\mathrm{PF}_{tt}^{\mathrm{NPSV}[k+1]} = \mathrm{PF}_{tt}^{\mathrm{NPSV}[k]}$. Then we have $\mathrm{P}_{tt}^{\mathrm{NPSV}[k+1]} = \mathrm{P}_{tt}^{\mathrm{NPSV}[k]}$. So, by Corollary 6.3, we have $\mathrm{P}_{tt}^{\mathrm{NP}[m]} = \mathrm{P}_{tt}^{\mathrm{NP}[k]}$ for every m. Since $\mathrm{P}^{\mathrm{NP}[k]} \subseteq \mathrm{NP}(k+1)$, BH collapses to its $(k+1)$-st level. □

COROLLARY 6.5. *If* $\mathrm{PF}^{\mathrm{NPSV}[k+1]} = \mathrm{PF}^{\mathrm{NPSV}[k]}$ *for some $k \geq 1$, then* BH *collapses to its 2^k-th level.*

Proof. Suppose $\mathrm{PF}^{\mathrm{NPSV}[k+1]} = \mathrm{PF}^{\mathrm{NPSV}[k]}$. Then we have $\mathrm{P}^{\mathrm{NPSV}[k+1]} = \mathrm{P}^{\mathrm{NPSV}[k]}$. So, by Corollary 6.2, we have $\mathrm{P}^{\mathrm{NP}[m]} = \mathrm{P}^{\mathrm{NP}[k]}$ for every m. Since $\mathrm{P}^{\mathrm{NP}[k]} \subseteq \mathrm{NP}(2^k)$, BH collapses to its 2^k-th level. □

The following proposition, which is the NPSV version of Theorem 3.5, requires a somewhat different proof from the NPMV case. It is interesting that our techniques seem insufficient to prove the reverse inclusion.

PROPOSITION 6.6. *For every $k \geq 1$,* $\mathrm{PF}^{\mathrm{NPSV}[k]} \subseteq \mathrm{PF}_{tt}^{\mathrm{NPSV}[2^k-1]}$.

Proof. Let $f \in \mathrm{PF}^{\mathrm{NPSV}[k]}$ via an oracle transducer M making k many queries to an oracle partial function $g \in$ NPSV computed by an NPSV machine N. We may assume that f is single-valued. Let r be the NPMV function defined by the machine U in the proof of Proposition 3.1, where $t(x) = k$. Since g is single-valued, it follows that r is single-valued also, and hence $r \in$ NPSV. Moreover, we have that $\lambda x.[r(x, 0^k)]$ is total and polynomial time computable, and if b_x is the lexicographically greatest string $b \in \Sigma^k$ on which $r(x, b)$ is defined, then

$$r(x, b_x) = \begin{cases} 0 & \text{if } x \notin \mathit{dom}(f), \\ 1y & \text{if } f(x) = y. \end{cases}$$

We compute $f(x)$ by querying r in parallel on the $2^k - 1$ values $\{(x, a) \mid a \neq 0^k\}$, recovering $f(x)$ from $r(x, a)$ as above, where a is lexicographically largest such that $r(x, a)$ returns a value. □

The equality in the following theorem depends heavily on the fact that NPSV functions are single-valued, and we do not believe it holds for oracles in NPMV. The first inclusion was found independently by E. Hemaspaandra [16].

THEOREM 6.7. $\mathrm{PF}^{\mathrm{NPSV}[\log]} \subseteq \mathrm{PF}_{tt}^{\mathrm{NPSV}} = \mathrm{PF}_{tt}^{\mathrm{NP}}$.

Proof. The first inclusion arises immediately from adapting the proof of Proposition 6.6. We now show that $\mathrm{PF}_{tt}^{\mathrm{NPSV}} \subseteq \mathrm{PF}_{tt}^{\mathrm{NP}}$. The reverse inclusion is obvious.

If h is any single-valued partial function, define $\mathit{code}(h)$ to be the set of all (x, i, b) such that the ith bit (left to right) of $h(x)$ exists and is b. Suppose $f \in \mathrm{PF}_{tt}^{\mathrm{NPSV}}$ is computed by a deterministic oracle transducer M running in time $p(n)$ and making parallel queries to an oracle $g \in$ NPSV, which itself is computed by a machine N

running in time $q(n)$ (p and q are polynomials). We may assume f is single-valued. Let g' be the function that maps x to $1y$ if $g(x)$ maps to y, and is undefined otherwise. Clearly, $\mathit{code}(g') \in \mathrm{NP}$, and since g' is single-valued, at most one of the tuples $(x, i, 0)$ and $(x, i, 1)$ is in $\mathit{code}(g')$, for all x and i.

Given an input x, we compute $f(x)$ by making parallel queries to $\mathit{code}(g')$ as follows: let $q_1, \ldots, q_s$ be the oracle queries made by $M(x)$. We query $\mathit{code}(g')$ on all the tuples (q_i, j, b) for $1 \leq i \leq s$, $0 \leq j \leq q(p(|x|))$, and $b \in \{0, 1\}$. Since $\mathit{dom}(g') = \mathit{dom}(g)$ and $g'(q)$ never maps to the empty string, $q_i \in \mathit{dom}(g)$ if and only if one of $(q_i, 0, 0)$ and $(q_i, 0, 1)$ is in $\mathit{code}(g')$. For each $q_i \in \mathit{dom}(g)$, we recover $g(q_i)$ in the usual way by reading from $\mathit{code}(g')$ all but the 0th bit of $g'(q_i)$. Query answers in hand, we continue simulating M to obtain $f(x)$. □

The next corollary was proved independently from scratch by L. Hemaspaandra [10].

COROLLARY 6.8. $\mathrm{P}_{tt}^{\mathrm{NPSV}} = \mathrm{P}_{tt}^{\mathrm{NP}} = \mathrm{P}^{\mathrm{NP[log]}} = \mathrm{P}^{\mathrm{NPSV[log]}}$.

Proof. The first equation follows from Theorem 6.7 by considering only characteristic functions. Also by Theorem 6.7, we have $\mathrm{P}^{\mathrm{NPSV[log]}} \subseteq \mathrm{P}_{tt}^{\mathrm{NP}} = \mathrm{P}^{\mathrm{NP[log]}} \subseteq \mathrm{P}^{\mathrm{NPSV[log]}}$, so the last equation holds. □

Recall from the Introduction that it is not known whether *sat* belongs to $\mathrm{PF}^{\mathrm{NPSV}[k]}$ for any k. We know that *maxsat* is complete for $\mathrm{PF}^{\mathrm{NPMV}}$ [13]. Thus, by Corollary 6.5, if, for any $k \geq 1$, $\mathit{maxsat} \in \mathrm{PF}^{\mathrm{NPSV}[k]}$, then the Boolean and polynomial hierarchies collapse.

Although $\mathrm{PF}^{\mathrm{NPMV[log]}} = \mathrm{PF}_{tt}^{\mathrm{NPMV}}$ (Theorem 3.13), we do not know whether $\mathrm{PF}^{\mathrm{NPSV[log]}}$ and $\mathrm{PF}_{tt}^{\mathrm{NPSV}}$ are equal. In particular, whereas, $\mathrm{PF}_{tt}^{\mathrm{NPSV}} = \mathrm{PF}_{tt}^{\mathrm{NP}}$ (Theorem 6.7) is easy to prove, apparently $\mathrm{PF}^{\mathrm{NPSV[log]}}$ and $\mathrm{PF}^{\mathrm{NP[log]}}$ are not equal, for $\mathrm{NPSV} \subseteq \mathrm{PF}^{\mathrm{NPSV[1]}} \subseteq \mathrm{PF}^{\mathrm{NPSV[log]}}$, while $\mathrm{NPSV} \subseteq \mathrm{PF}^{\mathrm{NP[log]}}$ implies $\mathrm{P} = \mathrm{UP}$ [15]. Thus, $\mathrm{PF}^{\mathrm{NPSV[log]}} \subseteq \mathrm{PF}^{\mathrm{NP[log]}}$ implies $\mathrm{P} = \mathrm{UP}$. Similarly, $\mathrm{PF}^{\mathrm{NPMV[1]}} \subseteq \mathrm{PF}^{\mathrm{NP[log]}}$ implies $\mathrm{P} = \mathrm{NP}$.

REFERENCES

[1] R. Beigel. NP-hard sets are P-superterse unless R = NP. Technical Report 88-04, Department of Computer Science, The Johns Hopkins University, 1988.

[2] R. Beigel. Bounded queries to SAT and the Boolean hierarchy. *Theor. Computer Science*, 84(2):199–223, 1991.

[3] H. Buhrman. 1992. Private communication.

[4] S. Buss and L. Hay. On truth table reducibility to SAT. *Information and Computation*, 91:86–102, 1991.

[5] R. Book, T. Long, and A. Selman. Quantitative relativizations of complexity classes. *SIAM J. Comput.*, 13(3):461–487, August 1984.

[6] J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The boolean hierarchy I: Structural properties. *SIAM J. Comput.*, 17(6):1232–1252, 1988.

[7] J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The boolean hierarchy II: Applications. *SIAM J. Comput.*, 18(1):95–111, 1989.

[8] J. Cai and L. Hemachandra. The Boolean hierarchy: Hardware over NP. In *Structure in Complexity Theory, Lecture Notes in Computer Science 223*, pages 105–124, Berlin, 1986. Springer-Verlag.

[9] Z. Chen and S. Toda. On the complexity of computing optimal solutions. *International Journal of Foundations of Computer Science*, 2:207–220, 1991.

[10] L. Hemaspaandra. 1993. Private communication.

[11] L. Hemaspaandra, A. Naik, M. Ogihara, A. Selman. Computing solutions uniquely collapses the polynomial hierarchy. *SIAM Journal on Computing*, in press.

[12] J. Kadin. The polynomial time hierarchy collapses if the Boolean hierarchy collapses. *SIAM Journal on Computing*, 17(6):1263–1282, December 1988.

[13] M. Krentel. The complexity of optimization problems. *J. Comput. Systems Sci.*, 36:490–509, 1988.

[14] J. Köbler, U. Schöning, and K. Wagner. The difference and truth-table hierarchies for NP. *Theoretical Informatics and Applications (RAIRO)*, 21:419–435, 1987.

[15] A. Selman. A taxonomy of complexity classes of functions. *J. Comput. Systems Sci.*, 48(2):357–381, 1994.

[16] E. Hemaspaandra. 1993. Private communication.

[17] H. Wareham. On the comptutational complexity of inferring evolutionary trees. Master's thesis, Department of Computer Science, Memorial University of Newfoundland, 1992.

[18] O. Watanabe and S. Toda. Structural analysis of the complexity of inverse functions. *Math. Systems Theory*, 26:203–214, 1993.

[19] G. Wechsung and K. Wagner. On the boolean closure of NP. In *Proc. International Conf. on Fundamentals of Computation Theory, Lecture Notes in Computer Science 199*, pages 485–493. Springer-Verlag, Berlin, 1985.